TP 4 : FILTRE PARTICULAIRE

Introduction

Le but du TP est d'implémenter un filtre particulaire pour suivre l'évolution d'un robot qui se déplace suivant une marche aléatoire discrète dans une pièce comportant des murs. À chaque instant, le robot choisit une direction au hasard, ainsi qu'une amplitude de déplacement. S'il rencontre un mur, il s'arrête juste avant le mur et passe au pas de temps suivant. L'expérimentateur dispose d'observations bruitées $Y_t, t \geq 1$ de la position du robot $X_t, t \geq 1$ et souhaite estimer sa position X_t à chaque instant, sachant toutes les observations passées. L'utilisation d'un modèle gaussien (même approximativement gaussien) n'est plus envisageable, en raison des arrêts forcés par les murs.

Pour simplifier, on discrétise le problème de la manière suivante (Figure 1). Le robot se déplace sur une 'grille', c'est à dire sur l'ensemble $[0, 1, \ldots, L-1, L]^2$, où $L \in \mathbb{N}^*$ est la taille de la grille. Les murs à l'intérieur de la pièce sont supposés seulement verticaux ou horizontaux. L'état du robot est un vecteur $X_t = (X_{t,1}, X_{t,2}) \in \{1, \ldots, L-1\}^2$ (le robot ne peut pas se trouver sur le bord).

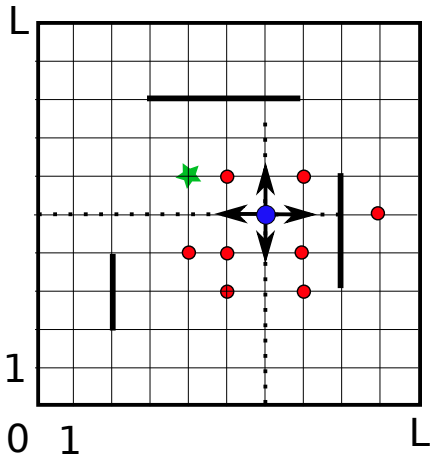

FIGURE 1 – Modèle du robot sur une grille avec 3 murs. Point bleu : état réel, 'caché'. Point vert : observation. Points rouges : particules du filtre. Traits noirs épais : murs.

- La position initiale du robot est tirée uniformément parmi les cases libres de la grille.
- Modèle d'évolution : Le robot choisit une des 4 directions au hasard (uniformément), et une longueur D de déplacement libre suivant une loi géometrique de paramètre θ_{state} (D est le nombre d'échecs avant succès dans une épreuve de Bernoulli de probabilité θ_{state}). Il se deplace alors de $\ell_t = \min(D, r)$ dans la direction choisie, où r est nombre de case maximum dans ladite direction telle que la trajectoire ne rencontre aucun mur ni le bord de la grille.
- Modèle d'observation : $Y_t = (Y_{t,1}, Y_{t,2})$ est la position 'bruitée' horizontalement et verticalement par deux lois géométriques indépendantes $\eta_{t,1}, \eta_{t,2}$ de paramètre commun θ_{obs}. Le bruit s'écrit alors $\eta_t = (S_{1,t}\eta_{1,t}, S_{2,t}\eta_{2,t})$, où $S_{1,t}, S_{2,t} \in \{-1, 1\}$ sont les signes des

perturbations horizontales et verticales, indépendantes et suivant une loi de Bernoulli sur $\{-1, 1\}$ de paramètre $1/2$. L'observation est alors $Y_t = X_t + \eta_t$. Le bruit ne tient pas compte des limites du domaine ni des murs, autrement dit on peut obtenir une observation sur un mur ou hors de la grille.

Quatre fonctions sont fournies sur le site pédagogique dans le dossier `robot-fonctions.zip`. Elles permettent de simuler l'évolution du robot, de générer des observations, et d'afficher le tout. Le dossier contient également la fonction `geornd` pour simuler des variables aléatoires de loi géométrique.

1 Simulation de la trajectoire et des observations

1. Le script ci-dessous appelle la fonction `robot_grille_simul` en appendice et permet de visualiser étape par étape la trajectoire et les observations. Faites tourner le script et prenez le temps de comprendre le rôle des paramètres modifiables.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Parametres modifiables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T = 200 ; %%horizon de temps
stepPlot = 10; %%, nombre de visualisations,
L = 20 %% taille de la grille
walls = [ 5,10, 10, 1 ;
      10, 3, 4, 0 ;
      3, 13, 4, 0;
      15, 13, 4, 0] %% murs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% parametres des lois geometriques des bruits (entre 0 et 1)
thetaState = 0.4
thetaObs = 0.3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if mod(T, stepPlot )~=0
  error(['total time T not divisible by the specified',...
       'number of plots stepPlot'])
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Generer (X,Y)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[X,Y] = robot_grille_simul(T, L, walls, thetaState, thetaObs)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Affichage
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all;
```

```
figure(1)
hold on ;
robot_grille_plot(L,walls, thetaObs);
plot(X(1,1), X(1,2), '*', 'color', 'b', 'markersize',5)
plot(Y(1,1), Y(1,2), '*', 'color', 'g', 'markersize',5)
timeLag = T / stepPlot ;
disp(['Let us see how the robot moves around (true state: blue ', ...
    'and how the observations look like (green)'])
for j = 0:(stepPlot-1)
  inds = (j * timeLag +(j==0 )) : ((j+1) * timeLag);
  hState= plot(X(inds,1), X(inds,2), 'o', 'color', 'b', ...
           'markersize', 8);
  hObs = plot(Y(inds,1), Y(inds,2), '*', 'color', 'g', ...
          'markersize', 8);
  hStateLine = plot(X(inds,1), X(inds,2), 'color', 'b');
  if j==0
    hleg= legend([hState, hObs,hStateLine], ...
              'true state',  'observations', 'recent trajectory');
  else
    hleg = legend([ hObs,hStateLine], ...
              'observations', 'recent trajectory');
  end
  disp('press any key')
  pause()
  delete(hleg)
  delete(hObs)
  delete(hStateLine)
end
hObs = plot(Y(:,1), Y(:,2), '*', 'color', 'g', 'markersize', 8 );
hStateLine = plot(X(:,1), X(:,2), 'color', 'b');
legend([ hObs,hStateLine], ...
     'observations', 'true trajectory')
hold off;
```

2 Filtrage particulaire

On implémente un filtre particulaire de type 'bootstrap filter'. Dans ce cas particulier, la loi de proposition pour la génération des particules est la loi d'évolution du modèle de l'état. Autrement dit, on fait évoluer les particules selon la loi du modèle (au cours de l'étape de proposition. On choisit de rééchantillonner les particules à intervalles de temps réguliers de taille `resample` $\in \mathbb{N}^*$.

1. Compléter le code de la fonction suivante, qui doit renvoyer un couple `[Xp,W]`, où `W` est le vecteur des poids et `Xp` est un tableau de dimension `(Np,2,T)` (nombre de particules * (abscisses, ordonnées) * horizon temporel).

```
function [ Xp, W] = particle_robot_grille(Y,L, walls, ...,
```

```
      thetaState, thetaObs, Np , resample)
%ARGUMENTS:
% Y, L, walls, thetaState, thetaObs: comme dans robot_grille_simul;
% Np : nombre de particules
%VALUE :
% Xp : tableau de taille ( Np, 2, T) (with T = size(Y,1) ), particules
% W : tableau de taille (Np, T ), poids.

%%%%%%%%%%%%%%%%%%%
%% Initialisation %
%%%%%%%%%%%%%%%%%%%
 T = size(Y,1) ; %% horizon de temps
 Xp = zeros(Np,2, T) ; %% tableau pour stocker les particules
 W = zeros(T, Np) ; %% tableau pour stocker les poids
 particles = zeros(Np,2) ;%% etat courant des particules
 logweights = zeros(1,Np); %% vecteur du logarithme des poids
%% Creation du jeu de particules
 for i = 1:Np
  [newwalls, x] = robot_grille_init(L, walls);
  particles(i,:) = x;
 end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Algorithme de filtrage:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 for k=1:T
    for i = 1:Np
     %% Faire evoluer les particules selon le modele
     particles(i,:) = ; % completer

     %% Mise a jour des poids (non normalises)
     %% COMPLETER
      logweights(i) = ; %%
    end
    %% normalisation des poids (cf echantillonnage d'importance)
    w = %% COMPLETER

    %% reechantillonnage a intervalles de temps reguliers
    %% de longeur 'resample'
    if mod(k,resample) == 0
    %% COMPLETER
     particles = ; %%
     w = ; %%
    end
    Xp(:,:,k) = particles;
    W(k,:) = w';
 end
end
```

Le script suivant vous permet de tester que votre fonction ne renvoie pas d'erreur

```
%% 1 - simulation de la trajectoire
%%% Parametres de la simulation
T = 200 ; % horizon de temps
stepPlot = 10; % nombre de visualisations,
L = 20 % taille de la grille
walls = [ 5,10, 10, 1 ;
```

```
      10, 3, 4, 0 ;
      3, 13, 4, 0;
      15, 13, 4, 0] % murs

thetaState = 0.6
thetaObs = 0.4

%%% parametres du filtrage
Np =10 % nombres de particules
stepPlot = 10;

if mod(T, stepPlot )~=0
  error(['total time T not divisible by the specified',...
       'number of plots stepPlot'])
end
%%% Simulation
[X,Y] = robot_grille_simul(T, L, walls, thetaState, thetaObs) ;

%% 2 - Filtrage
[Xp, W] = particle_robot_grille (Y, L, walls, ...,
                    thetaState, thetaObs, Np ,2);
```

2. Le couple renvoyé par la fonction `particle_robot_grille` permet de donner une valeur approchée de $\mathbb{E}(X_t|y_{1:t})$, l'espérance a posteriori de la position, sachant $y_{1:t}$. On déclare un tableau `Mp` contenant cette approximation (pour $t \in \{1, \dots, T\}$). Complétez pour cela le code suivant (à copier à la suite du script fourni à la question 1)

```
Mp = zeros(T,2);
for k = 1:T
   Mp(k,:) = %% COMPLETER
end
```

3. On vous fournit le script ci-dessous pour visualiser les résultats. Copiez-le à la suite des deux questions précédentes. Vous pouvez maintenant constater le comportement de votre filtre. Essayez-le en faisant varier les paramètres des lois géométriques des bruits, l'horizon temporel, la configuration de la pièce (les murs), le nombre de particules, ... A propos du nombre de particules : que se passe-t-il lorsqu'on utilise trop peu de particules ? trop de particules ?

```
%% Affichage de la trajectoire des particules
figure(2)
hold on ;
robot_grille_plot(L,walls, thetaObs);
plot(X(1,1), X(1,2), '*', 'color', 'b', 'markersize',5)
plot(Y(1,1), Y(1,2), '*', 'color', 'g', 'markersize',5)
timeLag = T / stepPlot ;
for j = 0:(stepPlot-1)
  lastIndex = (j+1) * timeLag;
  inds = (j * timeLag +(j==0 )) : lastIndex ;

  hState= plot(X(inds,1), X(inds,2), 'o', 'color', 'b', 'markersize', 8);
  hObs = plot(Y(inds,1), Y(inds,2), '*', 'color', 'g', 'markersize', 8);
  hStateLine = plot(X(inds,1), X(inds,2), 'color', 'b');
  hMeanFilter = plot(Mp(inds,1), Mp(inds,2), 'color', 'r');
  hparticle = plot(Xp(:,1,lastIndex) , Xp(:,2,lastIndex), ...
```

```
    '*', 'color', 'r', 'markersize', 15) ;
  hlastState = plot(X(lastIndex,1) , X(lastIndex,2), ...
    '*', 'color', 'b', 'markersize', 15) ;

  disp('press any key')
  pause()

  delete(hMeanFilter)
  delete(hparticle)
  delete(hlastState)
end
hold off
figure(3)
hold on;
robot_grille_plot(L,walls, thetaObs);
indplot = floor(linspace(1,T,T/2));

hStateLine = plot(X(indplot,1), X(indplot,2), 'color', 'b');
hMeanFilter = plot(Mp(indplot,1), Mp(indplot,2), 'color', 'r');
hold off;
```

A Appendice : Fonctions pour la simulation de la trajectoire du robot et des observations

A.1 Fonction principale

```
function [X,Y] = robot_grille_simul(T, L, walls, thetaState, thetaObs)
  %% ARGUMENTS
  % T: nombre d'iterations
  % L: taille de la grille (0:L * 0:L, avec murs en 0 et en L)
  % walls: matrice K * 4 : K : nombre de murs,
  %       colonnes: (abscisse 1, ordonnee 1, longueur, orientation)
  %       orientation = 1 pour horizontal, 0 pour vertical
  % thetastate: parametre de la loi geometrique gouvernant
  %           l' equation d'etat
  % thetaObs : idem pour l' equation observation
  %%%%%%%
  % Le robot choisit une des 4 directions au hasard, une longueur de
  % deplacement suivant une loi geometrique et se deplace du nombre de
  % case maximum dans ladite direction telle que sa trajectoire ne
  % rencontre aucun mur ni le bord de la grille;
  % Observation : la position 'bruitee' d'une loi geometrique selon
  % une des 4 directions. Ne tient pas compte des limites du domaine
  % ni des murs
  %%%%%%%
  %% VALEUR X, Y: matrices T* 2
  %     (abscisses et ordonnees des etats et des observations)

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %Initialisation
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  [walls, x] = robot_grille_init(L, walls);
  X = zeros(1,2);
  %%%%%%%%%%%%%%%%%%%%%%%%%% boucle 'temporelle' %%%%%%%%%%%%
  for k = 1:T
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% evolution du robot
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    x = robot_grille_stateStep(x, walls, thetaState) ;
    X(k,:) = x ;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% observation
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*
    Sign = discrete_rnd( 2, [-1, 1], 1/2 * ones(1,2) ) ;
    %% [-1 ou 1, -1 ou 1]
    noise= geornd(2, thetaObs) ;
    Y(k,:)= x + Sign .* noise ;
  end
end
```

A.2 Initialisation

```
function [newWalls, x] = robot_grille_init(L, walls)
%ARGUMENTS : cf robot_grille_simul.
%VALEUR:
% newWalls : murs augmentes incluant les frontieres de la grille
% x : position initiale

%% tirer uniformement la position initiale, en dehors des murs
  on_a_wall = true;
  while on_a_wall
    x = discrete_rnd(2, 1:(L-1) , 1/(L-1) * ones(1,L-1) );
    on_a_wall_vect = (x(1) >= walls(:,1)).* (x(1) <= walls(:,1) + ...
    walls(:,3) ) .* ...
     (x(2) == walls(:, 2) ) .* (walls(:,4) ==1) ;
         %% mur horizontal
    on_a_wall_vect = on_a_wall_vect + ...
     (x(2) >= walls(:,2) ) .* (x(2) <= walls(:,2) + ...
    walls(:,3) ) .* ...
     (x(1) == walls(:,1)) .* (walls(:,4) ==0) ;
                    %% mur vertical
    on_a_wall = any(on_a_wall_vect) ;
  end
  %% ajout des murs frontiere de la grille
  newWalls = [walls ; [0,0,L, 1 ;
                      0,0,L,0;
                      0, L, L, 1;
                      L,0, L, 0]];
end
```

A.3 Evolution de l'état (un pas de temps)

```
function xnew = robot_grille_stateStep(x,walls, thetaState)
% ARGUMENTS
% x : etat courant
% walls: matrice K * 4
% thetaState: parametre de la loi geometrique de l'etat
% VALEUR
% xnew: nouvelle position

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%% evolution du robot
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  nstate = 1;
  direction = discrete_rnd(nstate, 1:4, 1/4 * ones(1,4));
  %direction : (1 2 3 4) <-> (gauche bas droite haut)

  step = geornd(nstate, thetaState);
  orient = mod(direction, 2); %% horizontal <-> 1, vertical <-> 0
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%%% impact possible sur un mur perpendiculaire au deplacement
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  candidat_walls_indices = (walls(:,4) ~= orient ) .* ...
                     (walls(:,orient+1 ) <= x(orient+1)).* ...
```

```
                    ( (walls(:,orient+ 1 ) + walls(:,3 ) ) >= ...
                      x(orient + 1)) ;
  %% mur vertical / horizontal
  %% ordonnee/abscisse basse inferieure a la position
  %% ordonnee/abscisse haute superieure a la position ;

  candidat_walls = walls(logical(candidat_walls_indices), :) ;

  %% 2 - orient =1 : horizontal ; 2-orient ==2 : vertical
  distance_walls = candidat_walls(:,2 - orient) - x(2 - orient) ;

  if( (direction ==3) || (direction ==4)) %% vers la droite ou le haut
    dmin_abs = min(distance_walls( distance_walls > 0 ) );
  %% (minimum absolute distance)
  else %%vers la gauche ou le bas
    dmin_abs = min(- distance_walls( distance_walls < 0 ) );
  end
  is_impact = ( step >= dmin_abs) ;

  if(is_impact)
    step = dmin_abs -1 ;
  end
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  %%% impact possible sur un mur parallele au deplacement
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  candidat_walls_indices = ...
  (walls(:,4) == orient ) .* (walls(:,orient+1 ) == x(orient+1)) ;
  %% mur vertical / horizontal; orient = 0 pour vertical
  %% ordonnee/abscisse egale a la position
  candidat_walls = walls(logical(candidat_walls_indices), :) ;

  if( (direction ==3) || (direction ==4)) %% vers la droite ou le haut
    %% 2 - orient =1 : horizontal ; 2-orient ==2 : vertical
    distance_walls = candidat_walls(:,2 - orient) - x(2 - orient) ;
    dmin_abs = min(distance_walls( distance_walls > 0 ) );
  else
    distance_walls = candidat_walls(:,2 - orient) + ...
                candidat_walls(:,3) - x(2 - orient) ;
    dmin_abs = min(- distance_walls( distance_walls < 0 ) );
  end
  is_impact = (~isempty(dmin_abs) && step >= dmin_abs );

    if(is_impact)
      step = dmin_abs -1 ;
    end

    negative_incr = (direction <= 2 ) ;
    xnew = x;
    xnew(2 - orient ) = x(2- orient) + (-1)^(negative_incr) * step ;
end
```

A.4 Affichage

```
function robot_grille_plot(L, walls, thetaObs )
  marg = 2/thetaObs;
```

```
  nwalls = size(walls,1) ;
  axis([-marg, L+marg, -marg, L+marg], 'manual' )
  plot(0:L, 0 * ones(1,L+1), 'linewidth', 4, 'color', 'k')
  plot( 0:L, L * ones(1,L+1), 'linewidth', 4,'color', 'k' )
  plot( 0 * ones(1,L+1), 0:L, 'linewidth', 4,'color', 'k' )
  plot( L * ones(1,L+1), 0:L, 'linewidth', 4,'color', 'k' )
  for i = 1:nwalls
    if (walls(i,4) ==1 ) %%horizontal
      line([walls(i,1), walls(i,1) + walls(i,3)], ...
          [walls(i,2), walls(i,2)], ...
          'linewidth', 4)
    else
      line([walls(i,1), walls(i,1) ], ...
        [walls(i,2), walls(i,2)+ walls(i,3)], ...
          'linewidth', 4)

  end
end
  val =true;
end
```