Chaîne de Markov : deux problèmes d'arrêt optimal

Bernard Lapeyre
CERMICS, École des Ponts ParisTech

March 18, 2019

Contents

Les programmes suivant on été prévu pour être exécutés à l'aide `Scicoslab`. Ce programme peut être téléchargé librement sur www.scicoslab.org. Il est disponible sur MacOSX, Windows ou Linux (Debian, Ubuntu ou Fedora). Noter toutefois qu'il faut installer le server `X`, `XQuartz` sur MacOSX. Ces programmes peuvent aussi fonctionner, moyennant de légère modification, avec la version officielle de `Scilab`.
Le fichier source en `Scilab` correspondant à ce document est arret_scilab_ddi.sci. Il est partiellement mais pas totalement corrigé. Le correction complète arret_scilab_ddi_corrige.sci sera accessible à la fin du TD.

1 Le calcul du prix d'une option "américaine"

Rappel : le cas européen (calcul d'espérance)

On considère le modèle de Cox-Ross :

$$X_0 = x_0, X_{n+1} = X_n \left(d\mathbf{1}_{\{U_{n+1}=P\}} + u\mathbf{1}_{\{U_{n+1}=F\}} \right).$$

On choisit les valeurs numériques de la façon suivante

$$x_0 = 100, r_0 = 0,1, \sigma = 0,3.$$

et l'on définit p, r, u et d en fonction de N de la façon suivante :

$$p = 1/2,\ r = r_0/N,\ u = 1 + \frac{\sigma}{\sqrt{N}} \ \text{ et } \ d = 1 - \frac{\sigma}{\sqrt{N}}.$$

On cherche à évaluer $\mathbf{E}(f(N, X_N))$ où

$$f(N,x) = \frac{1}{(1+r)^N} \max(K - x, 0),$$

avec $K = x_0 = 100$ et $r = 0,05$.

1. Calculer ces prix d'options européennes (call et put) se ramène à des calculs d'espérance d'une fonction d'une chaîne de Markov. On implémente ici la méthode de calcul d'espérance par "programmation dynamique".

```
// Payoff du put
function res=gain_put(x,K) res=max(K-x,0);endfunction

// Payoff du call
function res=gain_call(x,K) res=max(x-K,0);endfunction

// Une fonction pour gérer le décalage de 1 dans les vecteurs
function res=inc(n) res=n+1;endfunction;

function res=prix_eu(x_0,N,gain)
  // Calcul du prix europeen a l instant 0
  // ATTENTION AU DECALAGE DE 1 EN TEMPS ET EN ESPACE
  U=zeros(inc(N),inc(N));
  for k=[0:N] do
    U(inc(N),inc(k))=gain(x_0*u^k*d^(N-k),K)/(1+r)^N;
  end;
  for n=[N-1:-1:0] do
    // le temps decroit de N-1 a 0
    U(inc(n),inc(0):inc(n))=p*U(inc(n+1),inc(1):inc(n+1))+ ...
                            (1-p)*U(inc(n+1),inc(0):inc(n));
  end;
  res=U(inc(0),inc(0));
endfunction;

function res=prix_eu_n(n,x_0,N,gain)
  // Calcul du prix a l instant n
  res=prix_eu(x_0,N-n,gain);
endfunction
```

2. On peut vérifier que lorsque $p = 1/2$ (ou $r = 0$) et $x = K$ le prix des puts et des calls coïncident. Pour le choix classique $(1 + r - d)/(u - d)$ (et $r \neq 0$), les prix sont différents, ce que l'on vérifie aussi.

```
function main_1()
  r_0=0.05;sigma=0.3;
  x_0=100;K=100;

  N=50;
  d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
  r=r_0/N;

  // Lorsque p=1/2 et K=x_0 le prix du call = le prix du put (exercice!)
  p=1/2;K=x_0;
```

```
  p_put=prix_eu_n(0,x_0,N,gain_put);
  p_call=prix_eu_n(0,x_0,N,gain_call);
  if abs(p_put-p_call) >= 1000*%eps then
    printf("WARNING: ces deux prix devrait coincider: %f &lt;&gt; %f\n",p_put,p_cal
  else
    printf("Parfait!\n");
  end

  p=(1+r-d)/(u-d);
  printf("Prix du call : %f\n",prix_eu_n(0,x_0,N,gain_call));
  printf("Prix du put : %f\n",prix_eu_n(0,x_0,N,gain_put));
endfunction
```

3. Voici un programme qui permet de calculer des histogrammes. Si `varargin` est vrai un dessin est tracé, sinon seul l'histogramme est retourné dans `H`.

```
function H=histo_discret(samples,maxsize,plot_flag)
  if exists('%nsp') then
    H=histo_discret_nsp(samples,maxsize,plot_flag);
  else
    H=histo_discret_scilab(samples,maxsize,plot_flag);
  end
endfunction

function H=histo_discret_scilab(samples,maxsize,plot_flag)
  // histogramme de tirages selon une loi discrète à valeurs entières
  // supposé prendre des valeurs entre 0 et max.
  H=0
  for k=0:maxsize do
    // Calcul du nbre de tirages valant k / Taille
    H(k+1)=length(find(samples==k)) ./size(samples,'*');
  end;

  // Si plot_flag=%f pas de dessin
  if plot_flag then
    xbasc;plot2d3(0:maxsize,H);
    f=gcf();
    Dessin=f.children(1).children(1).children(1);
    Dessin.thickness=10;Dessin.foreground=5;
  end;
endfunction

// version nsp
function H=histo_discret_nsp(samples,maxsize,plot_flag)
  if nargin <= 2 then plot_flag=%f;end
  // histogramme de tirages selon une loi discrête à valeurs entières
  // supposée prendre des valeurs entre 0 et max.
```

```
  // Si plot_flag=%f pas de dessin
  H=zeros(1,maxsize+1);
  [y,ind,occ]=unique(samples);
  H(y+1)=occ ./size(samples,'*');
  if plot_flag then
    xbasc;plot2d3(0:maxsize,H,line_color = 2,line_thickness = 10);
  end;
endfunction
```

Le cas américain (arrêt optimal)

On s'intéresse au cas d'un option américaine qui promet (en valeur actualisée en 0), si on l'exerce à l'intant n pour une valeur de X_n valant x

$$\frac{1}{(1+r)^n}\max(K-x,0).$$

On cherche à calculer son prix donné par

$$\sup_{\tau \text{t.a.} \leq N} \mathbf{E}(f(\tau, X_\tau)).$$

4. On pose $v(n,x) = (1+r)^n u(n,x)$, vérifier en utilisant l'équation qui définit u dans le cours, que v est solution de

$$\begin{cases} v(n,x) = \max\left[\dfrac{pv(n+1,xu)+(1-p)v(n+1,xd)}{1+r}, (K-x)_+\right], n < N, x \in E \\ v(N,x) = (K-x)_+, x \in E. \end{cases} \quad (1)$$

5. Ecrire un algorithme récursif (et inefficace ...) pour le calcul de v en recopiant l'équation (1)

```
function res=prix_recursif_am(x,k,N,gain)
  if k==N then res=gain(x);return;end;
  //QUESTION: res = <À COMPLÉTER>
  res=max(res/(1+r),gain(x));
endfunction

function res=prix_slow_am(x,N,gain)
  res=prix_recursif_am(x,0,N,gain);
endfunction
```

6. Ecrire un algorithme efficace de calcul de $v(n,x)$ (le prix de l'option américaine en n).

```
function res=prix_am(x_0,N,gain)
  // Calcul du prix americain a l instant 0
  U=zeros(N+1,N+1);
  x=x_0*u .^[0:N] .*d .^[N:-1:0];// les N+1 points de calcul en N
```

```
  U(N+1,1:N+1)=gain(x);// Valeur de U en N

  // ATTENTION AU DECALAGE DE 1 en espace et en temps
  for n=[N-1:-1:0] do
    // le temps decroit de N-1 a 0
    U(n+1,1:n+1)=p*U(n+2,2:n+2)+(1-p)*U(n+2,1:n+1);
    U(n+1,1:n+1)=U(n+1,1:n+1)/(1+r);// actualisation
    x=x_0*u .^[0:n] .*d .^[n:-1:0];// les points de calcul en n
    //QUESTION: U(n+1,1:n+1)= QUOI DE NOUVEAU POUR UNE OPTION AMERICAINE ?
  end;
  res=U(1,1);
endfunction;

function res=prix_am_n(n,x_0,N,gain)
  res=prix_am(x_0,N-n,gain);
endfunction
```

7. Pour $N = 10$, comparer les 2 méthodes pour vérifier que tout fonctionne. Effectuer des calculs de prix avec des N plus grands (uniquement pour la deuxième méthode, bien sûr).

```
function main_2()
  sigma=0.3;r_0=0.1;
  K=100;x_0=100;

  N=10;
  r=r_0/N;
  d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
  p=(1+r-d)/(u-d);//p=1/2;


  prix_am(x_0,N,gain_put)
  prix_slow_am(x_0,N,gain_put)

  // Les deux algos font ils le même chose ?
  // on verifie : prix_slow(x_0,N) \approx prix(x_0,N)
  p1=prix_slow_am(x_0,N,gain_put);
  p2=prix_am(x_0,N,gain_put);
  if (abs(p1-p2) >= 100*%eps) then
    printf("WARNING: ces deux prix devrait coincider: %f &lt;&gt; %f\n",p1,p2);
  else
    printf("Parfait!\n");
  end

  N=1000;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);

  prix_am(x_0,N,gain_put)
```

```
endfunction;
```

8. Tracer les courbes de prix américaines et européennes $x \to v(0,x)$ pour $x \in [80,120]$ et les supperposer au "payoff".

```
function main_3()
  // tracé de courbes
  N=50;
  sigma=0.3;
  p=1/2;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
  K=100;x_0=100;
  r_0=0.1;
  r=r_0/N;

  vmin=50;
  vmax=150;

  n=0;
  for x=[vmin:vmax] do
    n=n+1;
    //QUESTION: courbe_am(n) = <A COMPLÉTER>;
    //QUESTION: courbe_eu(n) = <A COMPLÉTER>;
  end
  // On compare les courbes "Américaines" et "Européennes"
  plot2d([vmin:vmax],courbe_eu,style = 2);
  plot2d([vmin:vmax],courbe_am,style = 3);
  plot2d([vmin:vmax],gain_put([vmin:vmax]),style = 4);
endfunction;
```

9. Regardez comment le prix évolue au fils du temps : tracez les courbes $x \to v(n,x)$, pour $n = 10, 20, 30, 40, 50$.

```
function main_4()
  sigma=0.3;
  K=100;x_0=100;

  N=50;
  p=1/2;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
  r_0=0.1;r=r_0/N;

  vmin=50;vmax=150;
  // évolution de la courbe en fonction de n
  for n=[0,10,20,30,40,45,47,49,50] do
    i=0;
    for x=[vmin:vmax] do
      i=i+1;
      //QUESTION: courbe_am(i) = <A COMPLÉTER>;
```

```
    end
    plot2d([vmin:vmax],courbe_am);
  end
endfunction
```

10. Que constatez vous lorsque N augmente ($N = 10, 100, 200, 500$) et que l'on choisit r, u et d en fonction de N comme définis plus haut.

Commentaire: lorsque N vers $+\infty$ dans ces conditions, on converge vers le prix dans un modèle continu (et célèbre : le modèle de Black et Scholes). Dans le cas européen, le résultat se prouve grâce au théorème de la limite centrale. Dans le cas américain, c'est un peu plus compliqué!

```
function main_5()
  // Avec cet algorithme on peut augmenter N
  // mais il faut renormaliser convenablement u et d.
  // Essayer avec N=10,100,200,...,1000

  sigma=0.3;
  K=100;x_0=100;

  r_0=0.1;
  p=1/2;
  N=50;

  // La renormalisation convenable
  // pour converger vers un modèle de Black et Scholes
  r=r_0/N;
  d=1-sigma/sqrt(N);
  u=1+sigma/sqrt(N);

  vmin=50;vmax=150;
  for N=[5,10,20,30,50] do
    r=r_0/N;
    d=1-sigma/sqrt(N);
    u=1+sigma/sqrt(N);
    n=0;
    for x=[vmin:vmax] do
      n=n+1;
      courbe(n)=prix_am(x,N,gain_put);
    end
    plot2d([vmin:vmax],courbe);
    plot2d([vmin:vmax],max(K-[vmin:vmax],0));// Le payoff ou l'obstacle
  end
endfunction
```

11. Ecrire une fonction qui simule le vecteur $(Bin(1), \ldots, Bin(N))$ du nombre de pile cumulé dans des N tirages à pile ou façe $(p, 1-p)$.

    Par la suite le valeur $X(n)$ du modèle de Cox-Ross à l'instant n s'écrira :

$$X(n) = x_0 u^{Bin(n)} d^{n-Bin(n)}.$$

```
function res=simul_bin(N,p)
  // sommes partielles nombre de N tirages a pile ou face (p,1-p)
  bin=grand(1,N,"bin",1,p);// tirages a pile ou face (p,1-p)
  //QUESTION: res=<A COMPLÉTER>;// sommes partielles
endfunction;
```

12. Calculer la prix américain en conservant dans un vecteur `V(t,k)` les valeurs en l'instant $t-1$ et au point $X(n) = x_0 u^{k-1} d^{n-(k-1)}$. Le décalage de 1 est du, comme d'habitude, au fait que les tableaux sont indicés à partir de 1 en ScicosLab.

```
function V=prix_am_vect(x_0,N,gain)
  // On calcule les valeurs de "v(n,x)" (voir question précédente)
  // mais, ici, on les conservent dans un vecteur "V"
  // ce qui évitera d'avoir à les re-calculer un grand
  // nombre de fois

  // ATTENTION AU DECALAGE DE 1 en ESPACE ET EN TEMPS
  V=zeros(N+1,N+1);
  x=x_0*u .^[0:N] .*d .^[N:-1:0];// les points de calcul en N
  V(N+1,1:N+1)=gain(x);

  for n=[N-1:-1:0] do
    // le temps décroit de N-1 a 0
    V(n+1,1:n+1)=p*V(n+2,2:n+2)+(1-p)*V(n+2,1:n+1);
    V(n+1,1:n+1)=V(n+1,1:n+1)/(1+r);// actualisation
    x=x_0*u .^[0:n] .*d .^[n:-1:0];// les points de calcul en  n
    V(n+1,1:n+1)=max(V(n+1,1:n+1),gain(x));
  end;
endfunction;
```

13. A partir d'une trajectoire du modèle de Cox-Ross donnée par le vecteur Bin et des valeurs de V précédemment calculées, évaluer le temps d'arrêt optimal associé à cette trajectoire.

```
function res=temps_optimal_option(Bin,V,gain)
  // Calcule le temps d'arrêt optimal associé à la trajectoire
  // |£(X(1),\ldots,X(N))£| où  |£X(n) =£|  x_0*u^Bin(n)*d^(n-Bin(n))
  // Bin est la somme cumulée de tirages de Bernouilli indépendants
  if gain(x_0)==V(1,1) then res=0;return;end;
  for n=1:size(Bin,'*')-1 do
    x=x_0*u .^Bin(n)*d .^(n-Bin(n));// Valeur de X(n)
    //QUESTION: if (ÉCRIRE LA CONDITION D'ARRET OPTIMAL & (gain(x) > 0)
```

```
    //              then res=n;return; end;
  end;
  res=N;
endfunction
```

14. Simuler un grand nombre de trajectoires du modèle de Cox-Ross et les valeurs des temps d'arrêt associés à ces trajectoires. Tracer un histogramme de la loi du temps d'arrêt optimal. Faire varier les valeurs de x_0 ($x_0 = 60, 70, 80, 100, 120$)) et voir l'influence de ce choix sur le loi du temps d'arrêt optimal.

```
function tau=un_test(x_0,N)
  sigma=0.3;
  p=1/2;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
  K=100;
  r_0=0.1;
  r=r_0/N;

  V=prix_am_vect(x_0,N,gain_put);
  Nbre=1000;
  for j=1:Nbre do
    Bin=simul_bin(N,p);
    tau(j)=temps_optimal_option(Bin,V,gain_put);
  end
  histo_discret(tau,max(tau),%t);
endfunction;

function main_6()
  // un test simple "at the money" |£(K=x_0)£|
  N=100;
  x_0=100;tau=un_test(x_0,N);
endfunction;

function main_7()
  N=50;
  x_0=60;tau=un_test(x_0,N);// on exerce toujours en 0
  x_0=70;tau=un_test(x_0,N);// on n'exerce plus (jamais) en 0
  x_0=80;tau=un_test(x_0,N);// On exerce de plus en plus tard ...
  x_0=100;tau=un_test(x_0,N);
  x_0=120;tau=un_test(x_0,N);// Le plus souvent en N
endfunction;
```

15. Traiter le cas du call. On peut montrer qu'il ne s'exerce qu'à l'échéance. Le vérifier par simulation.

```
// Teste le cas du call |£(x-K)_+£|
// qui ne s'exerce jamais avant l'écheance |£N£|.
// Le fait que le prix eu du call soit toujours plus grand
```

```
// que l'obstacle permet de le prouver rigoureusement.

function main_8()
  function res=gain_call(x)
    // Payoff du call
    res=max(x-K,0);
  endfunction

  sigma=0.3;r_0=0.1;
  K=100;x_0=100;

  N=50;
  r=r_0/N;
  u=(1+r)*exp(sigma/sqrt(N));d=(1+r)*exp(-sigma/sqrt(N));
  p=1/2;//p= (1+r-d)/(u-d); // en principe pour Cox-Ross

  V_call=prix_am_vect(x_0,N,gain_call);
  Nbre=1000;
  for j=1:Nbre do
    Bin=simul_bin(N,p);
    tau(j)=temps_optimal_option(Bin,V_call,gain_call);
  end
  histo_discret(tau,max(tau),%t);// s'exerce toujours en N
endfunction;
```

16. Évaluer la probabilité d'exercice anticipé (i.e. $\mathbf{P}(\tau < N)$, si τ est le temps d'arrêt optimal. Vérifier par simulation que cette probabilité est strictement positive (pour le put) si $r > 0$.

```
function proba=compute_proba(x_0,N)
  // Proba que tau<N
  sigma=0.3;
  p=1/2;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);
  K=100;
  r_0=0.1;
  r=r_0/N;

  V=prix_am_vect(x_0,N,gain_put);
  Nbre=1000;
  for j=1:Nbre do
    Bin=simul_bin(N,p);
    tau(j)=temps_optimal_option(Bin,V,gain_put);
  end

  //QUESTION: proba=QUE VAUT CETTE PROBA;
endfunction;
```

```
function main_9()
  N=50;
  for x_0=[60,70,80,100,120] do
    printf("x_0=%f: %f\n",x_0,compute_proba(x_0,N));
  end
endfunction
```

17. Tracer la frontière d'exercice en fonction du temps.

```
function s=frontiere(V)
  // Calcule la frontière d'exercice t -> s(t) t\in[0,N]
  N=size(V);
  N=N(1)-1;
  s=0;
  for n=0:N do
    for j=[0:n] do
      x=x_0*u^j*d^(n-j);
      // printf("n=%d j=%d V=%f obt=%f\n",n,j, V(n+1,j+1), obstacle(x));
      if V(n+1,j+1) > gain_put(x) then
        s(n+1)=x_0*u^(j-1)*d^(n-j+1);break;
      end;
    end
  end;
endfunction

function main_10()
  N=1000;
  r_0=0.05;r=r_0/N;
  sigma=0.3;
  K=100;x_0=70;

  p=1/2;d=1-sigma/sqrt(N);u=1+sigma/sqrt(N);

  V=prix_am_vect(x_0,N,gain_put);
  front=frontiere(V);
  plot2d(1:size(front,'*'),front);
endfunction;
```

2 Un problème de recrutement

On reçoit, consécutivement, N candidats à un poste. Les circonstances m'imposent de décider tout de suite du recrutement (soit on recrute la personne que l'on vient de recevoir, soit on la refuse définitivement). On cherche à maximiser la proabilité de recruter le meilleur candidat. Quelle est la meilleure stratégie ?

On a vu en cours que l'on peut se ramener à $(S_1, \ldots, S_N)$ est une suite de variables aléatoires indépendantes de Bernouilli $1/n$, qui est une chaîne de Markov sur l'espace

$E = \{0, 1\}$, non homogène, de matrice de transition, dépendant du temps, P_n

$$\begin{array}{rcl} P_n(0,0) & = & P_n(1,0) = 1 - \frac{1}{n+1} = \frac{n}{n+1}, \\ P_n(0,1) & = & P_n(1,1) = \frac{1}{n+1}. \end{array}$$

On chercher à maximiser $\mathbf{P}(\tau \text{ est le meilleur}) = \mathbf{E}\left(\frac{\tau}{N} S_\tau\right)$ parmi tous les temps d'arrêt.

1. Ecrire un programme `Scicoslab` qui calcule la solution $(u(n, 0 \text{ ou } 1), 0 \leq n \leq N)$ de l'équation de programmation dynamique donnée par (voir cours)

$$\begin{cases} u(n,x) = \max\left\{\frac{n}{n+1} u(n+1,0) + \frac{1}{n} u(n+1,1), \frac{n}{N} x\right\}, n < N, \\ u(N,x) = x, \end{cases}$$

   On calcule $(u(n, \{0, 1\}), 0 \leq n \leq N)$ à l'aide de l'équation de programmation dynamique. On désigne l'"obstacle" par $(f(n, \{0, 1\}), 0 \leq n \leq N)$. Notez que $f(n, 0) = 0$ et $f(n, 1) = n/N$.

```
function u=compute_u(N)
  u=zeros(N,2);
  u(N,:)=[0,1];
  for n=[N-1:-1:1] do
    temp=(n/(n+1))*u(n+1,1)+(1/(n+1))*u(n+1,2);
    // temp >= 0, donc u(n,0)=max(temp,0)=temp
    u(n,:)=[temp,max(temp,n/N)];
  end;
endfunction;
```

2. Tracer les courbes $u(n, 0)$, $u(n, 1)$ ainsi que "l'obstacle" $\frac{n}{N}$.

```
function main_11()
  N=1000;
  obstacle=[1:N]/N;

  u=compute_u(N);
  plot2d(1:N,obstacle,style = 2);
  plot2d(1:N,u(:,2),style = 3);// u(n,1)
  plot2d(1:N,u(:,1),style = 4);// u(n,0)
endfunction
```

3. Interprétez $u(0, 1)$ comme la probabilité de choisir le meilleur candidat avec une stratégie optimale ? Tracer la courbe N donne

$$\mathbf{P}(\text{Le candidat choisi par une stratégie optimale est le meilleur}),$$

   pour $N = 10, 25, 50, 100, 250, 500, 1000$. Vérifier numériquement qu'elle converge vers $1/e \approx 37\%$.

```
function main_12()
  // u(1,2) = P(succés pour la stratégie optimale)
```

```
  // On vérifie que cette proba tends vers 1/e = 37\%
  i=0;
  courbe=0;
  valeurs=[10,25,50,100,250,500,1000];
  for N=valeurs do
    u=compute_u(N);
    i=i+1;courbe(i)=u(1,2);
  end
  plot2d(valeurs,courbe)
endfunction
```

4. Vérifier que l'on a $u(n,0) > 0$ pour tout $n < N$ et donc, que $0 = f(n,0) \neq u(n,0)$. Par ailleurs, on a bien sûr, $u(N,0) = 0 = f(N,0)$. Comme le temps d'arret optimal τ est donné par

$$\tau = \inf\{n \geq 0, u(n,S_n) = f(n,S_n)\},$$

en déduire qu'il sera toujours postérieur à τ_{min}, le premier temps où $u(n,1) = f(n,1) = n/N$, (puisque $u(n,0) > 0 = f(n,0)$, sauf lorsque $n = N$) et que c'est la premier instant après τ_{min} pour lequel $S_n = 1$, si $n < N$.

Il découle de ce qui précéde qu'un temps d'arrêt optimal est donné par le premier instant après n_{min} (n_{min} compris) où $S_n = 1$ (ou encore $R_n = 1$, c'est à dire le premier instant où le nouvel arrivant est meilleur que ceux qui l'on précédé).

Ecrire un algorithme de calcul de ce temps τ_{min}.

```
function res=temps_min(N)
  // Calcule le
  // premier temps où |£u(n,1)=f(n,1)=n/N£|
  // à |£\epsilonf£| près.
  epsilon=10*%eps;
  u=compute_u(N);
  for n=[1:N] do
    if (abs(u(n,2)-(n/N)) < epsilon) then
      break;
    end
  end
  res=n;
endfunction
```

Vérifier par simulation que le temps (déterministe) τ_{min} "vaut environ" N/e pour N grand. On peut le démontrer sans trop de difficulté à l'aide de la série harmonique.

```
function main_13()
  // Vérification que temps_min vaut environ |£N/e£|.
  Taille=1000;
  x=0;
  for N=[1:Taille] do
    x(N)=temps_min(N)/N;
```

```
  end
  x=x-1/exp(1);
  nb=size(x,'*');
  plot2d(1:nb,x);
endfunction
```

5. Ecrire une fonction qui calcule la suite des rangs d'insertion (R dans le cours) d'une permutation et une fonction qui calcule la permutation définie par ses rangs d'insertion successifs.

```
function R=Omega2R(omega)
  // Calcule les rangs d'insertion R pour un |£\omega£| donné
  R=zeros(1,length(omega));
  for n=[1:length(omega)] do
    // classe le vecteur omega(1:n) en croissant
    y=gsort(omega(1:n),'g','i');
    // calcule l'indice de omega(n) dans le tableau classe y
    // c'est à dire son classement parmis les n premiers
    R(n)=find(omega(n)==y);
  end
endfunction

function omega=R2Omega(R)
  // Calcule |£\omega£| connaissant les rangs d'insertion

  N=length(R);
  temp=zeros(1,N);
  for n=[1:N] do
    // On place le n-ième individu à sa place: la R(n)-ième
    temp=[temp(1:R(n)-1),n,temp(R(n):n-1)];
  end;
  // On obtient une application |{\tt temp}| qui au rang associe l'individu.
  // Mais  |£\omega£|, c'est l'application qui à l'individu asscocie son rang:
  // c'est l'inverse de |{\tt temp}| qui se calcule par
  for n=[1:N] do omega(temp(n))=n;end;
  // ou si l'on est à l'aise en Scilab:
  // omega(temp)=1:N;//!!
endfunction

function main_14()
  // test sur une permutation
  N=10;
  omega=grand(1,'prm',[1:N]')';// tirage d'une permutation aléatoire.
  R=Omega2R(omega);
  // Retrouve t'on omega ?
  and(R2Omega(R)==omega)
endfunction
```

6. Vérifier par simulation que la probabilité d'obtenir le meilleur candidat, lorsque l'on utilise la stratégie optimale, est de l'ordre de $1/e \approx 37\%$. Ce qui n'est pas génial, mais c'est le mieux que l'on puisse faire (sans connaitre l'avenir!).

```
function n=temps_optimal(omega,tau_min)
  // Calcule le temps d'arret optimum
  // pour la permutation |£\omega£|

  // Calcule la suite S à partir de omega (omega->R->S)
  S=(Omega2R(omega)==1);

  // Le temps optimal se situe après |£\tau_min£| et c'est le premier instant
  // où |£S(n)=1£| apres ce temps, sauf si |£n=N£|, auquel cas on est oblige
  // de prendre  le dernier candidat.
  for n=[tau_min:N] do
    //QUESTION: if CONDITION D'ARRET OPTIMAL then break; end;
  end
  // Si on sort de cette boucle avec n=N et N est bien le temps optimal.
endfunction
```

On teste dans un cas particulier.

```
function main_15()
  // Verification que la probabilité d'obtenir
  // le meilleur est de l'ordre de 1/e
  N=100;
  Taille=10000;
  tau_min=temps_min(N);

  ss=0;
  for i=[1:Taille] do
    omega=grand(1,'prm',[1:N]');
    tirages(i)=temps_optimal(omega,tau_min);

    // tirages(i) est il le meilleur ?
    if (omega(tirages(i))==1) then ss=ss+1;end;
  end
  proba=ss/Taille;
  printf("probabilité d''obtenir le meilleur: %.3f\n",proba);

  // histo_discret(tirages,N);
endfunction
```