Réseau de Communication avec Débordement

Xavier CHASSAGNEUX
Rodolphe GOURSEAU

3 Juillet 2002

Résultats pour une ligne téléphonique isolée

Réponse à la question de simulation n° 1

La simulation est contenue dans la sous-section *Projet de simulation* de la section *Programmes de simulation*. Le calcul de $\frac{T_t(k)}{T_t(0)}$ est donné par *SimulTel(* ρ, C, T_{max} *)* , où T_{max} est le temps final de la simulation.

Les résultats montrent que pour $\rho = 0.9$ peu d'appels sont rejetés, voire aucun. Pour $C = 500$, comme $C - \lambda = 50$, on observe une légère saturation, car λ est trop proche de C . Pour $C = 1000$, comme $C - \lambda = 100$, l'écart entre λ et C est suffisamment grand pour qu'il n'y ait pas d'appels rejetés. Les courbes *1* et *2* en annexes illustrent ce propos.

Lorsque ρ croît, le poids de la courbe se déplace vers C . Le nombre d'appels rejetés est de plus en plus important, comme le montre l'évolution des courbes *1*, *3*, et *4* pour $C = 500$.

Réponse à la question théorique n° 2

Pour savoir quelle est l'occupation la plus fréquente, il faut calculer l'espérance de la variable aléatoire X_t (car d'après le préambule sur la loi invariante $\lim_{t\to\infty} \mathbf{P}(X_t \in \textit{circuit}\ k) = \pi(k)$).

On a alors :

$$\begin{aligned} & \mathbf{E}(X_t) &=& \sum_{k=1}^{C} k\,\pi(k) \\ \Rightarrow \quad & \mathbf{E}(X_t) &=& \sum_{k=1}^{C} k \left(\frac{\lambda^k}{k!} \left(\sum_{i=0}^{C} \frac{\lambda^i}{i!} \right)^{-1} \right) \\ \Rightarrow \quad & \mathbf{E}(X_t) &=& \lambda \frac{\displaystyle\sum_{k=0}^{C-1} \frac{\lambda^k}{k!}}{\displaystyle\sum_{i=0}^{C} \frac{\lambda^i}{i!}} \end{aligned}$$

En remarquant que $\lim_{n\to\infty} \sum_{j=0}^{n} \frac{\lambda^j}{j!} = e^{\lambda}$, on obtient :

$$\mathbf{E}(X_t) \underset{C\to\infty}{\longrightarrow} \lambda \qquad (\, car \;\; \lambda \leq C \,)$$

Réponse à la question de simulation n° 2

La simulation est contenue dans la sous-section *Projet de simulation* de la section *Programmes de simulation*. Le calcul de k , occupation la plus fréquente de la ligne, est donné par *AfficheResult*(ρ, C, T_{max}) .

La courbe est la représentation de $k = f(\lambda)$ pour $C = 1000$ et $400 \leq \lambda \leq 620$. L'allure de la courbe est celle de la première bissectrice ($k = \lambda$). Les fortes variations de part et d'autre de la première bissectrice sont dues au fait que $T_{max} = 20$ n'est pas assez élevé, mais nous n'avons pas pû prendre uune valeur supérieure car le temps de calcul était trop long.

Un réseau avec débordement

Réponse à la question théorique n° 3

politique n° 2

Dans l'expression de $T_t^{ab,2,n}$, le terme $\sum_{c\neq a,b} \mathbf{1}_{X_s^{ac,n}<C, X_s^{bc,n}<C}$ comptabilise le nombre de lignes libres entre les villes a et b via une troisième ville. Il y a n-2 villes pouvant jouer le rôle de la troisième ville, dons on introduit un $\frac{1}{n-2}$ pour "normaliser". En prenant $1 - \frac{1}{n-2} \sum_{c\neq a,b} \mathbf{1}_{X_s^{ac,n}<C, X_s^{bc,n}<C}$, on a ainsi la proportion (entre 0 et 1) de lignes occupées entre a et b via une troisième ville. Ainsi $T_t^{ab,2,n}$ est la proportion de temps où les appels entre a et b via une troisième ville sont bloqués. Mais d'après la propriété énoncée dans la première partie ($\lim_{t\to\infty} T_t = \pi(C) = R(\rho, C)$), on obtient :

$$\lim_{t\to\infty} T_t^{ab,2,n} = R_2^n(\rho, C)$$

politique n° 3

Dans l'expression de $T_t^{ab,3,n}$, le terme $1 - \mathbf{1}_{X_s^{ac,n}<C, X_s^{bc,n}<C}$ vaut 0 si la ligne entre a et b via c est libre et 1 si la ligne entre a et b via c est occupée. Alors, le produit $\prod_{c \neq a,b} \left\{1 - \mathbf{1}_{X_s^{ac,n}<C, X_s^{bc,n}<C}\right\}$ vaut donc 0 s'il existe une ligne libre entre a et b via une troisième ville, et vaut 1 si l'appel est perdu (suivant les critères de cette politique). Ainsi, on obtient :

$$\lim_{t\to\infty} T_t^{ab,3,n} = R_3^n(\rho, C)$$

Réponse à la question de simulation n° 4

politique n° 2

La simulation est contenue dans la sous-section *Projet de simulation avec debordement* de la section *Programmes de simulation*. Le calcul de différentes valeurs de $T_t^{ab,2,n}$ en fonction t est donné par *SimulTelP2(* ρ, C, n, T_{max}, ΔT *)* , où ΔT est l'écart entre deux mesures de $T_t^{ab,2,n}$.

On a choisi $T_{max} = 20$ pour la simulation. Pour $\rho = 0.9$, on constate que $T_t^{ab,2,n} = 0$. Il n'y a donc pas d'appels passant par une troisième ville. Pour $\rho = 0.95$, $T_t^{ab,2,n}$ est très faible. Le nombre d'appels passant par une troisième ville augmente plus rapidement quand n croît. Ainsi, plus de lignes sont occupées, et la proportion de temps où le système rejette les appels croît. Les courbes *7*, *8*, et *9* illustrent ce fait. Pour $\rho = 1$, $T_t^{ab,2,n}$ est rapidement non-nul et tend vers une valeur limite (non-nulle). La courbe *10* en donne un aperçu. Pour $\rho = 1.05$, $T_t^{ab,2,n}$ croît rapidement vers sa valeur limite (cf : courbe *11*).

politique n° 3

La simulation est contenue dans le fichier la sous-section *Projet de simulation avec debordement P3* de la section *Programmes de simulation*. Le calcul de différentes valeurs de $T_t^{ab,3,n}$ en fonction t est donné par *SimulTelP3(* ρ, C,

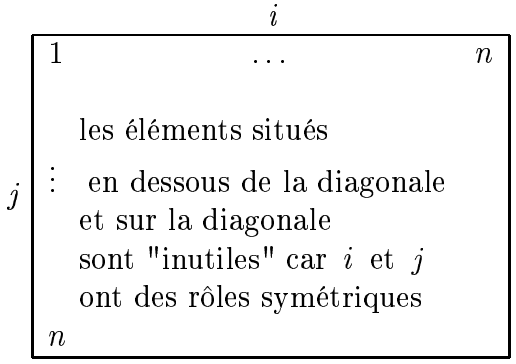
$n,\ T_{max}\ ,\ \Delta T\ )$, où ΔT est l'écart entre deux mesures de $T_t^{ab,3,n}$.

En analysant les résultats de la simulation, on arrive aux mêmes conclusions que celles de la politique 2. Cependant, le régime stationnaire est atteint plus rapidement pour la politique 3. Le nombre d'appels passant par une troisième ville est souvent plus grand pour cette politique. Les courbes *10* et *12* permettent de comparer le nombre d'appels passant par une troisième ville obtenue respectivement pour la politique 2 et pour la politique 3.

Programmes de simulation

Explication du système de numérotation pour passer d'un tableau à deux dimensions à un tableau à une dimension

Pour stocker les lignes entre les villes i et j , le plus simple est d'utiliser un tableau à deux dimensions (d'éléments $t_{i,j}$). Mais moins de la moitié du tableau est utilisée (car i et j ont des rôles symétriques), ce qui entraîne une perte de mémoire inutile et un ralentissement de la simulation.



Donc on numérote le tableau à deux dimensions de telle sorte que l'on n'ait plus qu'un tableau à une seule dimension de $\frac{n(n-1)}{2}$ éléments notés a_l .

$a_1 \quad a_2 \quad \dots \quad a_{n-1}$ $a_{n+1} \quad \dots \quad a_{2n-3}$ les éléments $\ddots$ situés en dessous de la diagonale et sur la diagonale ne sont pas comptabilisés	

Par récurrence, on montre que :

si $i, j \in \{1, n\}$ avec $i > j$, alors on a : $l = i - j + \dfrac{(2n-j)(j-1)}{2}$

On peut également passer du tableau à une dimension au tableau à deux dimensions (d'éléments $t_{i,j}$) grâce aux relations suivantes[1] :

$$j = \mathbb{E}\Big[\frac{2n+1-\sqrt{(2n-1)^2+8-8l}}{2}\Big]$$

$$i = j + l - \frac{(2n-j)(j-1)}{2}$$

(En effet, si l est à la ligne j, on a, comme $i > j$:

$$1 + \frac{(2n-j)(j-1)}{2} \leq l < 1 + \frac{(2n-j-1)(j)}{2} ,$$

donc la plus petite valeur de l est : $l = 1 + \dfrac{(2n-j)(j-1)}{2}$, et en développant, on a : $2 - 2l + 2nj - 2n + j - j^2 = 0$

$$\text{donc :} \quad \Delta = (2n-1)^2 + 8 - 8l \quad \text{et alors :} \quad j = \frac{2n+1 \pm \sqrt{\Delta}}{2}$$

Mais comme $j \leq n$ et que la fonction $j \mapsto -j^2 + (2n+1)j + 2 - 2l - 2n$ est croissante sur $]-\infty\,;\,\dfrac{2n+1}{2}[$, on obtient l'expression de j donnée ci-dessus, et on en déduit i.)

[1] $\mathbb{E}[\ ]$ est la fonction partie entière

Projet de simulation

Les commentaires du programme sont en italique.

```
function [f,Tt0]=SimulTel (rho,C,Tmax)      Tmax est le temps l'où on arrète la simulation
   NbAppel=0
   ligne=zeros(1,C)    simule les lignes s'il y a zéro alors pas d'appel sinon l'appel finit à t
   Tt=zeros(1,C)    en fait Tt*t
   Tt0=0
   TempsActuel=0
   NombreComEnCours=0
   lambda=C*rho
   while (TempsActuel<Tmax)
      ProchainAppel=TempsActuel+grand(1,1,'exp',1/lambda)
       while %T    recherche du prochain événement (fin d'un appel) avant le prochain appel
          Mini=ProchainAppel
          for i=1 :C
             if ((ligne(i)>0) & (ligne(i)<Mini)) then Mini=ligne(i) ;imin=i ; end
          end
          if (Mini==ProchainAppel) then break ; end    pas de fin d'appel avant le prochain
sinon fin de l'appel imin
          if (Mini>Tmax) then break ; end    pour ne pas dépasser le temps fixé
          Tt(NombreComEnCours)=Tt(NombreComEnCours)+Mini-TempsActuel
          NombreComEnCours=NombreComEnCours-1
          ligne(imin)=0
          TempsActuel=Mini
       end
Ajout d'un nouvel appel si possible
       if (ProchainAppel>Tmax) then    pour ne pas depasser le temps fixé
          if (NombreComEnCours==0)
             Tt0=Tt0+ Tmax - TempsActuel
          else
             Tt(NombreComEnCours)=Tt(NombreComEnCours)+ Tmax - TempsActuel
          end
          break
       end
       if (NombreComEnCours==C) then //echec de l'appel
          NbAppel=NbAppel+1
          Tt(C)=Tt(C)+ProchainAppel-TempsActuel
          TempsActuel=ProchainAppel
       else
recherche d'une ligne libre
          for i=1 :C
             if (ligne(i)==0) then lign=i ;break :end    lign est une ligne libre
          end
          if (NombreComEnCours==0) then
             Tt0=Tt0 + ProchainAppel-TempsActuel
          else
             Tt(NombreComEnCours)=Tt(NombreComEnCours) + ProchainAppel-TempsActuel
          end
          ligne(lign)=ProchainAppel+grand(1,1,'exp',1)    durée de l'appel aléatoire
          NombreComEnCours=NombreComEnCours+1
          NbAppel=NbAppel+1
          TempsActuel=ProchainAppel
       end
    end
    for i=1 :C
       f(i)=Tt(i)/Tt0
    end
endfunction
```

```
function [k] = AfficheResult(rho,C,Tmax)    k indice tel que l'occupation est la plus fréquente
    [a,b]=SimulTel(rho,C,Tmax)
    Max=0
    for i=1 :C
       if a(i)>Max then k=i ;Max=a(i) ;end
    end
    plot (a)
endfunction

function [] = AfficheResult2(Lambdad,pas,Lambdaf,C,Tmax)
affiche la courbe des k      j=1
    y=zeros(1,(Lambdaf-Lambdad)/pas+1)
    x=zeros(1,(Lambdaf-Lambdad)/pas+1)
    for l=Lambdad :pas :Lambdaf
       rho=l/C
       [a,b]=SimulTel(rho,C,Tmax)
       Max=0
       for i=1 :C
          if a(i)>=Max then k=i ;Max=a(i) ;end
       end
       y(j)=k
       x(j)=l
       j=j+1
    end
    plot (x,y)    affichage de la courbe obtenue par simulation
endfunction
```

Projet de simulation avec debordement

Les commentaires du programme sont en italique.

```
function [l]=deux_en_un (i,j,n)
    x=max(i,j)
    y=min(i,j)
    l=(x-y)+(2*n-y)*(y-1)/2
endfunction

function [x,y]=un_en_deux (l,n)
    delta=(2*n-1)^2+8*(1-l)
    y=int(((2*n+1)-sqrt(delta))/2)
    x=l+y-(2*n-y)*(y-1)/2
endfunction

function [l]=circuits(i,j,tab,n)    rend le tableau des C circuits joignant les villes i,j
    x=max(i,j)    utile pour ne pas utiliser un tableau à 2 dimensions
    y=min(i,j)    coûteux en espace mémoire
    l=tab((x-y)+(2*n-y)*(y-1)/2, :,1)    utile également pour le tableau d'occupation, rend alors le nombre
de cicuits utilisés entre i et j
endfunction

function [l]=occupation(i,j,tab,n)    Idem pour un tableau à une dimension (l'occupation)
    x=max(i,j)    rend alors le nombre de cicuits utilisés entre i et j
    y=min(i,j)
    l=tab((x-y)+(2*n-y)*(y-1)/2)
endfunction

function []=sauvecircuit (i,j,t,tab,n)    t est le nouveau tableau des circuits joignant les villes i et j
    x=max(i,j)
    y=min(i,j)
    tab((x-y)+(2*n-y)*(y-1)/2, :,1)=t
```

```
endfunction

function []=sauveoccupation (i,j,t,tab,n)    t est la nouvelle valeur de l'occupation entre les villes i et j
    x=max(i,j)
    y=min(i,j)
    tab((x-y)+(2*n-y)*(y-1)/2)=t
endfunction

function [res]=CalculIntP2(a,b,n,C,DureeEtat,TabOccupation)    calcul de int(0,t,1(X=C)*...)
    if (TabOccupation(deux_en_un(a,b,n))<C) then    DureeEtat : temps depuis lequel on est dans cet
état
       res=0
    else
       somme=0
       for i = 1 :n
          if (i<>a & i<>b) then
             if ((TabOccupation(deux_en_un(a,i,n))<C) & (TabOccupation(deux_en_un(i,b,n))<C)) then
somme=somme +1 ;end
          end
       end
       res = DureeEtat * (1- somme/(n-2))
    end
endfunction

function [TempsActuel,Tt,Tt2,Ttlast]=
ChangementEtat(TempsActuel,NouveauTemps,TabOccupation,Tt,n,C,Tt2,Ttlast)    fonction à exécuter
avant de changer le tableau d'occupation pour calculer Tt lorsque l'on change d'état (ajout ou suppres-
sion d'un appel)
    if (Ttlast+1)*DeltaT<NouveauTemps then    on ne refait le calcul des Tt que si une sauvegarde de
Tt2 va être faite
       for i=1 :(n^2-n)/2
          [a,b]=un_en_deux(i,n)
          Tt(i)=Tt(i)+CalculIntP2(a,b,n,C,NouveauTemps-TempsActuel,TabOccupation)
       end
    end
    while (Ttlast+1)*DeltaT<NouveauTemps    calcul des Tt2
       Ttlast=Ttlast+1
       for i=1 :(n^2-n)/2
          Tt2(i,Ttlast)=Tt(i)/(Ttlast*DeltaT)
       end
    end
    TempsActuel=NouveauTemps
endfunction
Pour l'appel : [TempsActuel,Tt,Tt2,Ttlast]=ChangementEtat(TempsActuel, ? ? ?,TabOccupation,Tt,n,C,Tt2,Ttlast)

function [Tt,Tt2]=SimulTelP2 (rho,C,n,Tmax,DeltaT)    Politique 2, C circuits par ligne n villes, DeltaT
est le temps entre chaque mesure pour Tt2
    ligne=zeros((n^2-n)/2,C,2)    simule les lignes par un tableau à 2 dimensions : s'il y a zéro alors pas
d'appel sinon l'appel finit à t
Pour le 3e indice : si 1 alors rend t, si 2 alors rend la troisième ville concernée par cet appel, si connexion
directe entre a et b avec a<b alors rend b
    Tt=zeros(1,(n^2-n)/2)    en fait Tt*t; on utilise deux_en_un() pour connaître Tt(ab,2,n)
    Tt2=zeros((n^2-n)/2,Tmax/DeltaT)    effectue une mesure de Tt2 tous les DeltaT
    Ttlast=0    dernière mesure effectuée pour les Tt2
    TabOccupation=zeros(1,(n^2-n)/2)
    ProchainDecrochage=zeros(1,(n^2-n)/2)    donne le prochain décrochage entre les villes a et b
    TempsActuel=0
    lambda=C*rho
    NbLiason=(n^2-n)/2
    for i=1 :NbLiason    remplissage du tableau du prochain décrochage
       ProchainDecrochage(i)= grand(1,1,'exp', 1/lambda)
    end
```

```
    while (TempsActuel<Tmax)
       Mini=ProchainDecrochage(1)
       imin=1
       for i=2 :NbLiason     recherche du prochain appel
          if ProchainDecrochage(i)<Mini then Mini=ProchainDecrochage(i) ;imin=i ;end
       end
       ProchainAppel=Mini
       for j=1 :NbLiason
          tempt=ligne(j, :,1)   ceci permet d'accélérer les calculs : le temps d'accès à un tableau à 3 di-
mensions est très long
          for i=1 :C
             if (tempt(i)>0) then
                if tempt(i)<Mini then Mini=tempt(i) ;imin=i ;jmin=j ; end
             end
          end
       end
       if (Mini==ProchainAppel) then     le prochain événement est un appel
Ajout d'un nouvel appel si possible
          if (TabOccupation(imin)==C) then     ligne a à b saturée, tentative par une autre ligne
             [a,b]=un_en_deux(imin,n)     a>b
             rand('uniform')
             temp=int(rand()*(n-2))+1     tirage aléatoire de la ville par où passer
             if (temp>=b) then
                temp=temp+1
                if (temp>=a) then temp=temp+1 ;end     pour que c≠b et c≠a
             end
             if (TabOccupation(deux_en_un(a,temp,n))==C|TabOccupation(deux_en_un(temp,b,n))==C)
then
                [TempsActuel,Tt,Tt2,Ttlast]=ChangementEtat(TempsActuel,ProchainDecrochage(imin),TabOccupation,Tt,n,C,Tt2,T
annulation de l'appel
                ProchainDecrochage(imin)=ProchainDecrochage(imin)+grand(1,1,'exp', 1/lambda)
             else   nouvel appel entre a et temp, et entre temp et b; recherche d'une ligne entre a et temp
                lign=0
                for i =1 :C
                   if ligne(deux_en_un(a,temp,n),i,1)==0 then lign=i ;break ;end
                end
                lign2=0     recherche entre temp et b
                for i =1 :C
                   if ligne(deux_en_un(b,temp,n),i,1)==0 then lign2=i ;break ;end
                end
                ligne(deux_en_un(a,temp,n),lign,1)=ProchainDecrochage(imin)+grand(1,1,'exp',1)
                ligne(deux_en_un(b,temp,n),lign2,1)=ligne(deux_en_un(a,temp),lign,1)
                ligne(deux_en_un(a,temp,n),lign,2)=b
                ligne(deux_en_un(b,temp,n),lign2,2)=a
                [TempsActuel,Tt,Tt2,Ttlast]=ChangementEtat(TempsActuel,ProchainDecrochage(imin),TabOccupation,Tt,n,C,Tt2,T
                TabOccupation(deux_en_un(a,temp,n))=TabOccupation(deux_en_un(a,temp,n))+1
                TabOccupation(deux_en_un(b,temp,n))=TabOccupation(deux_en_un(b,temp,n))+1
                ProchainDecrochage(imin)=ProchainDecrochage(imin)+grand(1,1,'exp',1/lambda)
             end
          else     ligne a à b non saturée; recherche d'une ligne libre
             for i=1 :C
                if (ligne(imin,i,1)==0) then lign=i ;break :end     lign est une ligne libre
             end
             ligne(imin,lign,1)=ProchainDecrochage(imin)+grand(1,1,'exp',1)
             [a,b]=un_en_deux(imin,n)     a>b
             ligne(imin,lign,2)=a     dans ce cas, on met max(a,b) dans ligne( , , 2)
[TempsActuel,Tt,Tt2,Ttlast]=ChangementEtat(TempsActuel,ProchainDecrochage(imin),TabOccupation,Tt,n,C,Tt2,Ttlast)
             ProchainDecrochage(imin)=ProchainDecrochage(imin)+grand(1,1,'exp',1/lambda)
             TabOccupation(imin)=TabOccupation(imin)+1
          end
       else   sinon fin de l'appel imin, jmin
jmin est l'entier représentant la ligne entre la ville a et b; imin est l'entier représentant le circuit de cette
```

```
ligne
        [a,b]=un_en_deux (jmin,n)
        [TempsActuel,Tt,Tt2,Ttlast]=ChangementEtat(TempsActuel,ligne(jmin,imin,1),TabOccupation,Tt,n,C,Tt2,Ttlast)
        if ligne(jmin,imin,2)<>a then
cas où la communication n'est pas établie directement entre a et b
            c=ligne(jmin,imin,2)
il existe deux possibilités : a↔b↔c ou c↔a↔b
            for i=1 :C
               if ligne(deux_en_un(b,c,n),i,1)==ligne(jmin,imin,1) then cas=1 ;lign=i ;end   cas b↔c
               if ligne(deux_en_un(a,c,n),i,1)==ligne(jmin,imin,1) then cas=2 ;lign=i ;end    cas a↔c
            end
            if cas==0 then disp("Erreur") ;end
            if cas==1 then   b↔c
               ligne(deux_en_un(b,c,n),lign,1)=0
               TabOccupation(deux_en_un(b,c,n))=TabOccupation(deux_en_un(b,c,n))-1
            else
               ligne(deux_en_un(a,c,n),lign,1)=0
               TabOccupation(deux_en_un(a,c,n))=TabOccupation(deux_en_un(a,c,n))-1
            end
         end
         ligne(jmin,imin,1)=0
         TabOccupation(jmin)=TabOccupation(jmin)-1
      end
   end
endfunction
```

Projet de simulation avec debordement P3

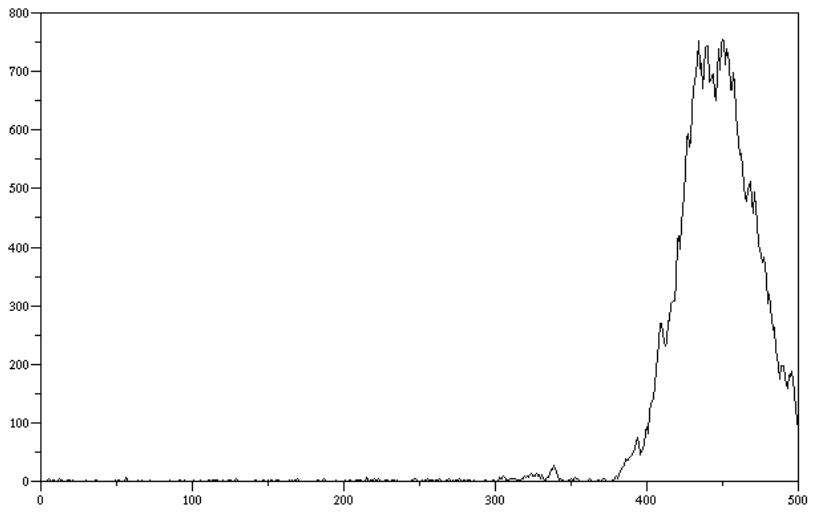
Les commentaires du programme sont en italique.

```
function [l]=deux_en_un (i,j,n)
   x=max(i,j)
   y=min(i,j)
   l=(x-y)+(2*n-y)*(y-1)/2
endfunction

function [x,y]=un_en_deux (l,n)
   delta=(2*n-1)^2+8*(1-l)
   y=int(((2*n+1)-sqrt(delta))/2)
   x=l+y-(2*n-y)*(y-1)/2
endfunction

function [l]=occupation(i,j,tab,n)    Idem pour un tableau à une dimension (l'occupation)
   x=max(i,j)    rend alors le nombre de cicuits utilisés entre i et j
   y=min(i,j)
   l=tab((x-y)+(2*n-y)*(y-1)/2)
endfunction

function [res]=CalculIntP3(a,b,n,C,DureeEtat,TabOccupation)    calcul de int(0,t,1(X=C)*...)
   if (TabOccupation(deux_en_un(a,b,n))<C) then     DureeEtat : temps depuis lequel on est dans cet
état
      res=0
   else
      res=DureeEtat
      for i = 1 :n
         if (i<>a & i<>b) then
            if ((TabOccupation(deux_en_un(a,i,n))<C) & (TabOccupation(deux_en_un(i,b,n))<C)) then
res=0 ;break ;end
         end
```

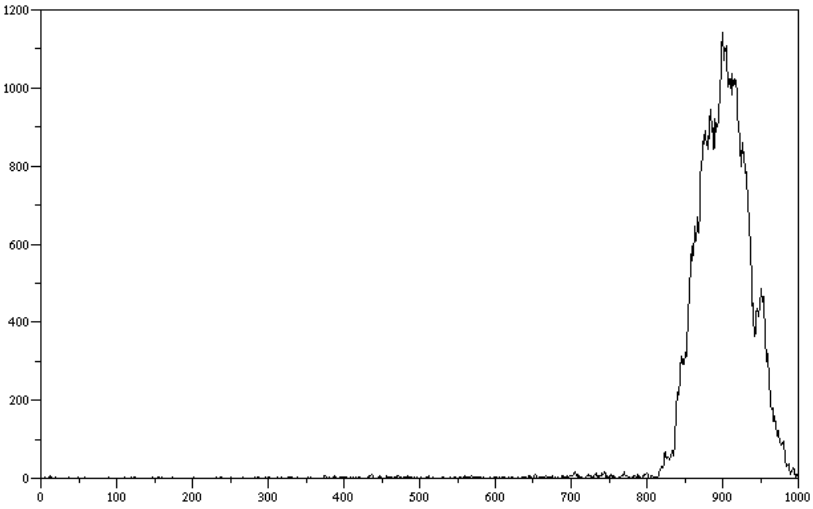
```
      end
    end
endfunction

function [TempsActuel,Tt,Tt2,Ttlast]=ChangementEtatP3(TempsActuel,NouveauTemps,TabOccupation,Tt,n,C,Tt2,Ttlast)
fonction à exécuter avant de changer le tableau d'occupation pour calculer Tt lorsque l'on change d'état
(ajout ou suppression d'un appel)
    if (Ttlast+1)*DeltaT<NouveauTemps then    on ne refait le calcul des Tt que si une sauvegarde de
Tt2 va être faite
       for i=1 :(n^2-n)/2
          [a,b]=un_en_deux(i,n)
          Tt(i)=Tt(i)+CalculIntP3(a,b,n,C,NouveauTemps-TempsActuel,TabOccupation)
       end
    end
    while (Ttlast+1)*DeltaT<NouveauTemps    calcul des Tt2
       Ttlast=Ttlast+1
       for i=1 :(n^2-n)/2
          Tt2(i,Ttlast)=Tt(i)/(Ttlast*DeltaT)
       end
    end
    TempsActuel=NouveauTemps
endfunction
Pour l'appel : [TempsActuel,Tt,Tt2,Ttlast]=ChangementEtatP3(TempsActuel,? ? ?,TabOccupation,Tt,n,C,Tt2,Ttlast)

function [Tt,Tt2]=SimulTelP3 (rho,C,n,Tmax,DeltaT)   Politique 3, C circuits par ligne n villes, DeltaT
est le temps entre chaque mesure pour Tt2
    ligne=zeros((n^2-n)/2,C,2)    simule les lignes par un tableau à 2 dimensions : s'il y a zéro alors pas
d'appel sinon l'appel finit à t
Pour le 3e indice : si 1 alors rend t, si 2 alors rend la troisième ville concernée par cet appel, si connexion
directe entre a et b avec a<b alors rend b
    Tt=zeros(1,(n^2-n)/2)    en fait Tt*t; on utilise deux_en_un() pour connaître Tt(ab,2,n)
    Tt2=zeros((n^2-n)/2,Tmax/DeltaT)    effectue une mesure de Tt2 tous les DeltaT
    Ttlast=0    dernière mesure effectuée pour les Tt2
    TabOccupation=zeros(1,(n^2-n)/2)
    ProchainDecrochage=zeros(1,(n^2-n)/2)    donne le prochain décrochage entre les villes a et b
    TempsActuel=0
    lambda=C*rho
    NbLiason=(n^2-n)/2
    for i=1 :NbLiason    remplissage du tableau du prochain décrochage
       ProchainDecrochage(i)= grand(1,1,'exp', 1/lambda)
    end
    while (TempsActuel<Tmax)
       Mini=ProchainDecrochage(1)
       imin=1
       for i=2 :NbLiason    recherche du prochain appel
          if ProchainDecrochage(i)<Mini then Mini=ProchainDecrochage(i) ;imin=i ;end
       end
       ProchainAppel=Mini
       for j=1 :NbLiason
          tempt=ligne(j, :,1)    ceci permet d'accélérer les calculs : le temps d'accès à un tableau à 3 di-
mensions est très long
          for i=1 :C
             if (tempt(i)>0) then
                if tempt(i)<Mini then Mini=tempt(i) ;imin=i ;jmin=j ; end
             end
          end
       end
       if (Mini==ProchainAppel) then    le prochain événement est un appel
Ajout d'un nouvel appel si possible
          if (TabOccupation(imin)==C) then    ligne a à b saturée, tentative par une autre ligne
             [a,b]=un_en_deux(imin,n)    a>b
             temp=0
```

```
            for i=1 :n
               if (i<>a & i<>b) then
                  if (TabOccupation(deux_en_un(a,i,n))<>C & TabOccupation(deux_en_un(i,b,n))<>C)
then
                     temp=i
                  end
               end
               if (temp<>0) then break ;end
            end
            if temp==0 then   annulation de l'appel
[TempsActuel,Tt,Tt2,Ttlast]=ChangementEtatP3(TempsActuel,ProchainDecrochage(imin),TabOccupation,Tt,n,C,Tt2,Ttlast)
               ProchainDecrochage(imin)=ProchainDecrochage(imin)+grand(1,1,'exp', 1/lambda)
            else   nouvel appel entre a et temp, et entre temp et b; recherche d'une ligne entre a et temp
               lign=0
               for i =1 :C
                  if ligne(deux_en_un(a,temp,n),i,1)==0 then lign=i ;break ;end
               end
               lign2=0    recherche entre temp et b
               for i =1 :C
                  if ligne(deux_en_un(b,temp,n),i,1)==0 then lign2=i ;break ;end
               end
               ligne(deux_en_un(a,temp,n),lign,1)=ProchainDecrochage(imin)+grand(1,1,'exp',1)
               ligne(deux_en_un(b,temp,n),lign2,1)=ligne(deux_en_un(a,temp),lign,1)
               ligne(deux_en_un(a,temp,n),lign,2)=b
               ligne(deux_en_un(b,temp,n),lign2,2)=a
[TempsActuel,Tt,Tt2,Ttlast]=ChangementEtatP3(TempsActuel,ProchainDecrochage(imin),TabOccupation,Tt,n,C,Tt2,Ttlast)
               TabOccupation(deux_en_un(a,temp,n))=TabOccupation(deux_en_un(a,temp,n))+1
               TabOccupation(deux_en_un(b,temp,n))=TabOccupation(deux_en_un(b,temp,n))+1
               ProchainDecrochage(imin)=ProchainDecrochage(imin)+grand(1,1,'exp',1/lambda)
            end
         else   ligne a à b non saturée; recherche d'une ligne libre
            for i=1 :C
               if (ligne(imin,i,1)==0) then lign=i ;break :end    lign est une ligne libre
            end
            ligne(imin,lign,1)=ProchainDecrochage(imin)+grand(1,1,'exp',1)
            [a,b]=un_en_deux(imin,n)    a>b
            ligne(imin,lign,2)=a    dans ce cas, on met max(a,b) dans ligne( , , 2)
[TempsActuel,Tt,Tt2,Ttlast]=ChangementEtatP3(TempsActuel,ProchainDecrochage(imin),TabOccupation,Tt,n,C,Tt2,Ttlast)
            ProchainDecrochage(imin)=ProchainDecrochage(imin)+grand(1,1,'exp',1/lambda)
            TabOccupation(imin)=TabOccupation(imin)+1
         end
      else   sinon fin de l'appel imin, jmin
jmin est l'entier représentant la ligne entre la ville a et b; imin est l'entier représentant le circuit de cette
ligne
         [a,b]=un_en_deux (jmin,n)
         [TempsActuel,Tt,Tt2,Ttlast]=ChangementEtatP3(TempsActuel,ligne(jmin,imin,1),TabOccupation,Tt,n,C,Tt2,Ttlast)
         if ligne(jmin,imin,2)<>a then
cas où la communication n'est pas établie directement entre a et b
            c=ligne(jmin,imin,2)
il existe deux possibilités : a↔b↔c ou c↔a↔b
            for i=1 :C
               if ligne(deux_en_un(b,c,n),i,1)==ligne(jmin,imin,1) then cas=1 ;lign=i ;end   cas b↔c
               if ligne(deux_en_un(a,c,n),i,1)==ligne(jmin,imin,1) then cas=2 ;lign=i ;end    cas a↔c
            end
            if cas==0 then disp("Erreur") ;end
            if cas==1 then   b↔c
               ligne(deux_en_un(b,c,n),lign,1)=0
               TabOccupation(deux_en_un(b,c,n))=TabOccupation(deux_en_un(b,c,n))-1
            else
               ligne(deux_en_un(a,c,n),lign,1)=0
               TabOccupation(deux_en_un(a,c,n))=TabOccupation(deux_en_un(a,c,n))-1
            end
```

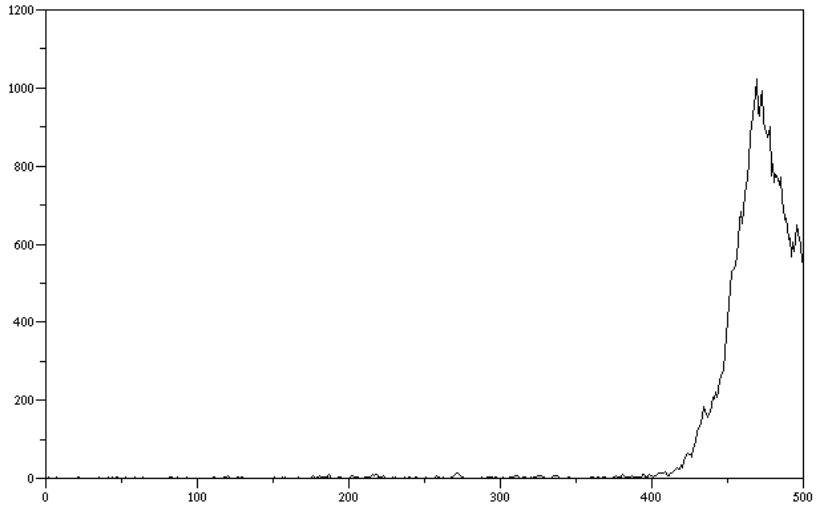
```
            end
            ligne(jmin,imin,1)=0
            TabOccupation(jmin)=TabOccupation(jmin)-1
        end
    end
endfunction
```



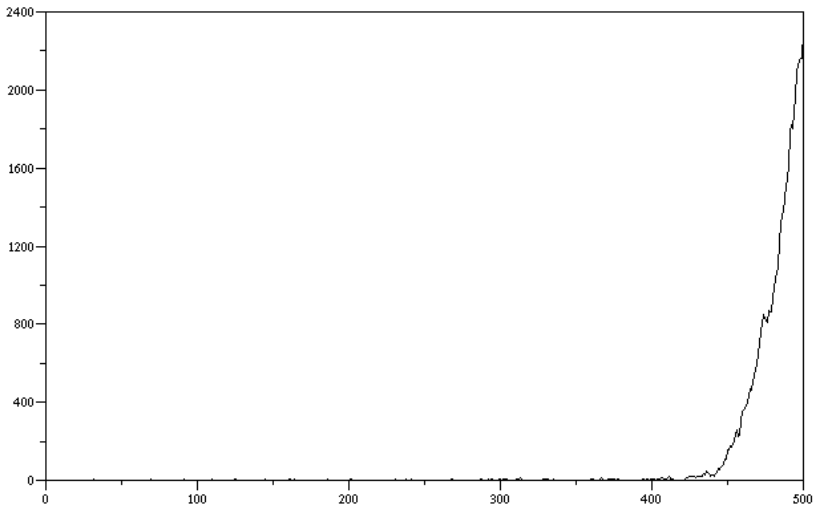
courbe *1* : $\dfrac{T_t(k)}{T_t(0)} = f(k)$: $\rho = 0.9 \quad C = 500 \quad T_{max} = 100$



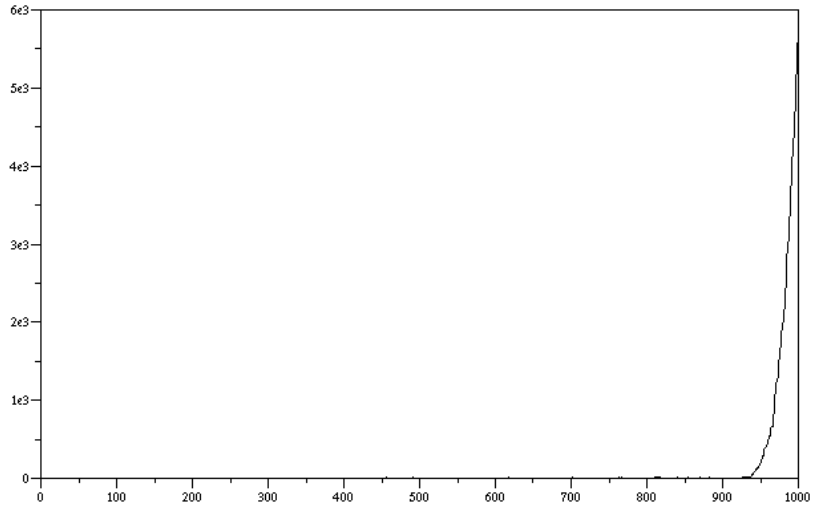
courbe *2* : $\dfrac{T_t(k)}{T_t(0)} = f(k)$: $\rho = 0.9 \quad C = 1000 \quad T_{max} = 100$



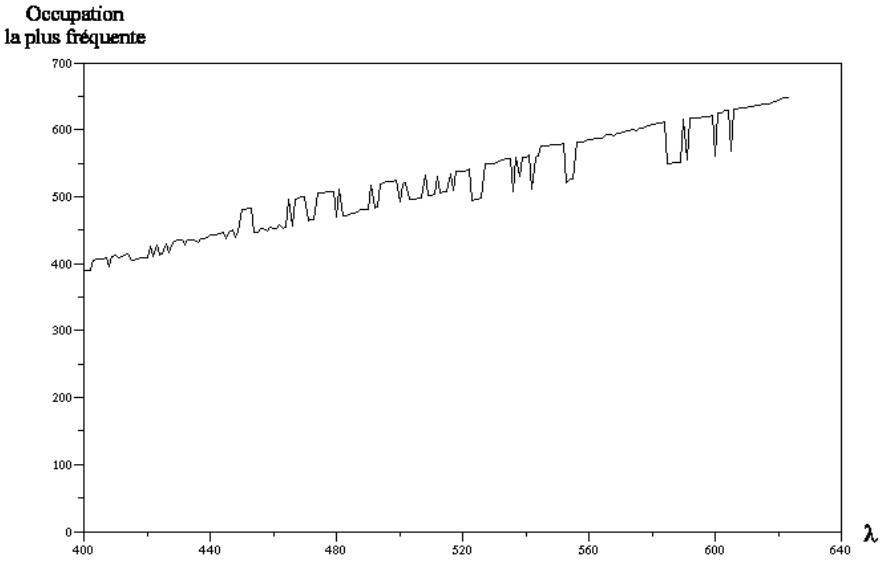
courbe *3* : $\dfrac{T_t(k)}{T_t(0)} = f(k) : \ \rho = 0.95 \quad C = 500 \quad T_{max} = 100$



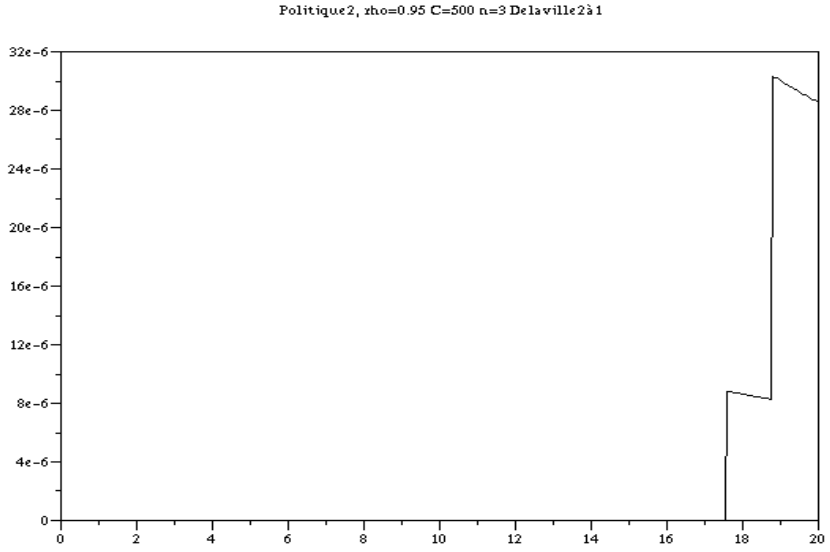
courbe *4* : $\dfrac{T_t(k)}{T_t(0)} = f(k) : \ \rho = 1 \quad C = 500 \quad T_{max} = 100$



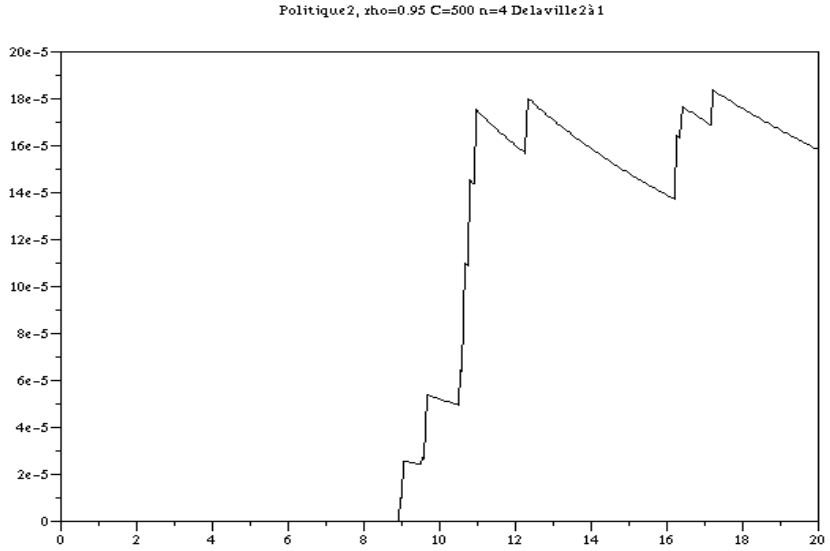
courbe *5* : $\dfrac{T_t(k)}{T_t(0)} = f(k) : \ \rho = 1 \quad C = 1000 \quad T_{max} = 100$



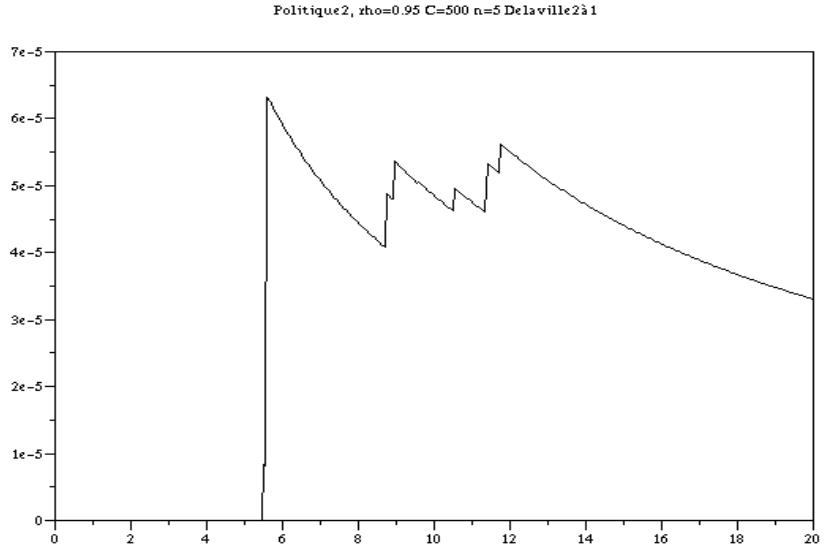
courbe *6* : $k = f(\lambda) : \ 400 \leq \lambda \leq 620 \quad C = 1000$



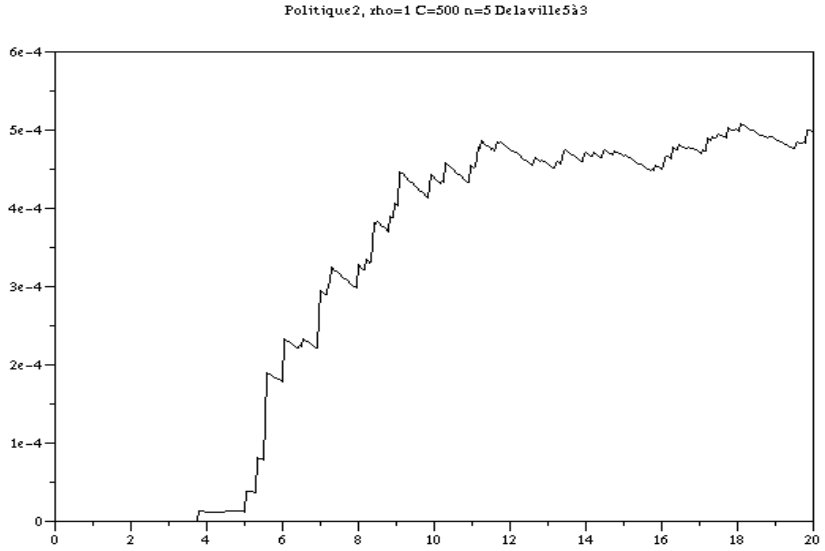
courbe *7* : $T_t^{ab,2,n} = f(t)$: $\rho = 0.95 \quad C = 500 \quad n = 3 \quad T_{max} = 20$



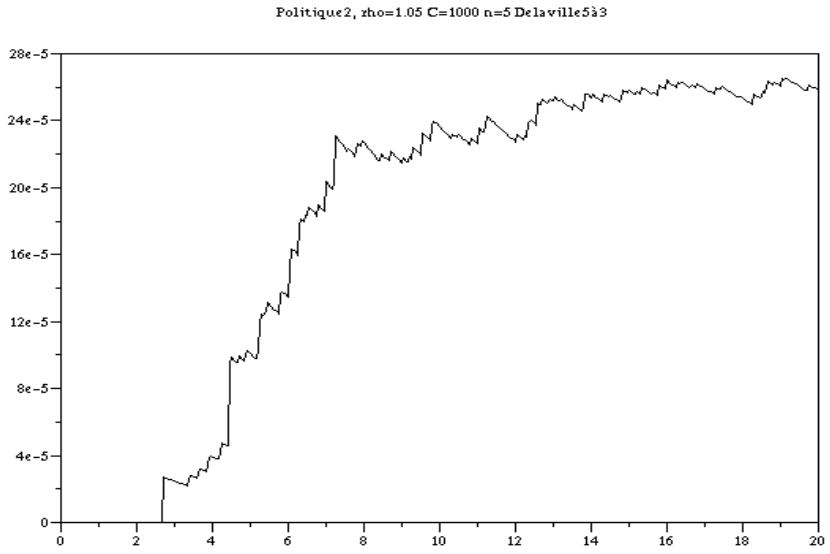
courbe *8* : $T_t^{ab,2,n} = f(t)$: $\rho = 0.95 \quad C = 500 \quad n = 4 \quad T_{max} = 20$



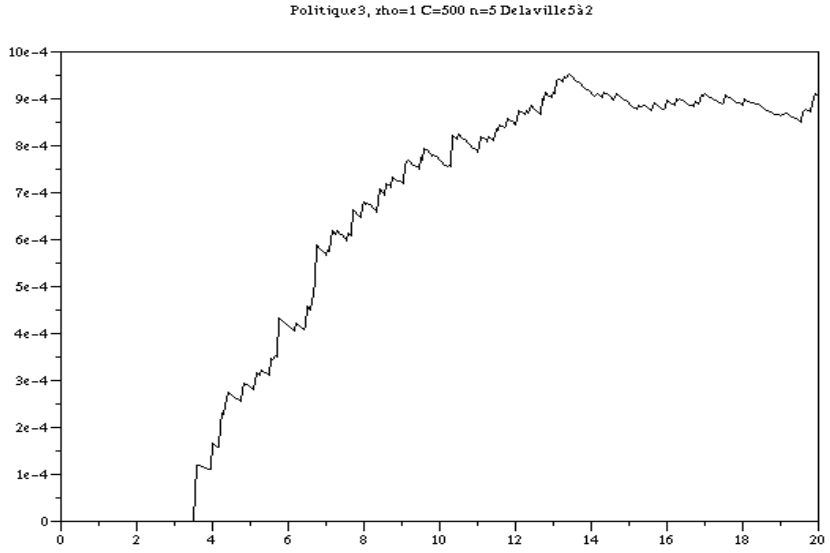
courbe *9* : $T_t^{ab,2,n} = f(t)$: $\rho = 0.95 \quad C = 500 \quad n = 5 \quad T_{max} = 20$



courbe *10* : $T_t^{ab,2,n} = f(t)$: $\rho = 1 \quad C = 500 \quad n = 5 \quad T_{max} = 20$



courbe *11* : $T_t^{ab,2,n} = f(t) : \ \rho = 1.05 \quad C = 1000 \quad n = 5 \quad T_{max} = 20$



courbe *12* : $T_t^{ab,3,n} = f(t) : \ \rho = 1 \quad C = 500 \quad n = 5 \quad T_{max} = 20$