UNIVERSITÉ DE GENÈVE
Sections d'Informatique et de Physique

FACULTÉ DES SCIENCES
Profs. B. Chopard et M. Droz

Outils, modélisation et simulation en calcul numérique – Corrigé série 12

7 juin 2005

Exercice 1.
Programme Fortran 90 :

```
module fonctions
implicit none
integer, parameter :: chosenpattern=1 !chosen pattern
real, parameter :: noise=0.9 !amplitude of the noise
real, parameter :: temperature=0.05 !temperature for Monte-Carlo
integer, parameter :: L1=32 !system size x
integer, parameter :: L2=32 !system size y
integer, parameter :: M=2 !total # patterns (see subroutine "store")
integer, parameter :: Nsteps=20 !# Monte-Carlo steps
character*(*), parameter :: filename = 's12.txt'
integer :: seed
contains
!---------------------------------------------------------------------
FUNCTION ran2(idum) !Initialize idum with negative integer.
INTEGER idum,IM1,IM2,IMM1,IA1,IA2,IQ1,IQ2,IR1,IR2,NTAB,NDIV
REAL :: ran2,AM,EPS,RNMX
PARAMETER (IM1=2147483563,IM2=2147483399,AM=1./IM1,IMM1=IM1-1,IA1=40014,&
          &IA2=40692,IQ1=53668,IQ2=52774,IR1=12211,IR2=3791,NTAB=32,&
          &NDIV=1+IMM1/NTAB,EPS=1.2e-7,RNMX=1.-EPS)
INTEGER :: idum2,j,k,iv(NTAB),iy
SAVE iv,iy,idum2
DATA idum2/123456789/, iv/NTAB*0/, iy/0/
if(idum.le.0)then
  idum=max(-idum,1)
  idum2=idum
  do j=NTAB+8,1,-1
    k=idum/IQ1
    idum=IA1*(idum-k*IQ1)-k*IR1
    if(idum.lt.0)idum=idum+IM1
    if(j.le.NTAB)iv(j)=idum
  enddo
  iy=iv(1)
endif
k=idum/IQ1
idum=IA1*(idum-k*IQ1)-k*IR1
if(idum.lt.0)idum=idum+IM1
k=idum2/IQ2
idum2=IA2*(idum2-k*IQ2)-k*IR2
if(idum2.lt.0)idum2=idum2+IM2
j=1+iy/NDIV
iy=iv(j)-idum2
iv(j)=idum
if(iy.lt.1)iy=iy+IMM1
ran2=min(AM*iy,RNMX)
return
```

```
end function ran2
!-----------------------------------------------------------------------
subroutine store(p) !initial patterns
integer, dimension(1:L1*L2,1:M), intent(inout) :: p
integer :: i
p=-1
p((L2/2-2)*L1+1:(L2/2+1)*L1,1)=1 !horizontal line
do i=0,L2-1 !vertical line
  p(i*L2+L1/2-1:i*L2+L1/2+1,2)=1
end do
return
end subroutine store
!-----------------------------------------------------------------------
subroutine hebb(w,p)
real, dimension(1:L1*L2,1:L1*L2), intent(inout) :: w
integer, dimension(1:L1*L2,1:M), intent(in) :: p
integer :: i,j,k
w=0.
do i=1,L1*L2
  do j=1,L1*L2
    do k=1,M
      w(i,j)=w(i,j)+real(p(i,k)*p(j,k))
    end do
  end do
  w(i,i)=0.
end do
w=w/real(L1*L2)
return
end subroutine hebb
!-----------------------------------------------------------------------
subroutine initial(s0,p) !initial pattern with noise (diffusion)
integer, dimension(1:L1*L2), intent(in) :: p
integer, dimension(1:L1*L2), intent(inout) :: s0
integer, dimension(1:L1*L2) :: tmp
real :: prob
integer :: i,j,to
tmp=p
do j=1,1+int(noise*real(min(L1,L2))) !number of diffusion steps
  s0=tmp
  do i=1,L1*L2
    if (tmp(i)==1) then
      if (ran2(seed) < noise) then !diffusion of black square
        prob=ran2(seed)
        if (prob<0.25) then !to the left
          to=i-1;if (to==0) to=L1*L2
          if (s0(to)==-1) then; s0(to)=1;s0(i)=-1; end if
        else if ((prob >0.25) .and. (prob<0.5)) then !to the right
          to=i+1;if (to==L1*L2+1) to=1
          if (s0(to)==-1) then; s0(to)=1;s0(i)=-1; end if
        else if ((prob >0.5) .and. (prob<0.75)) then !up
          to=i-L1;if (to<1) to=(L2-1)*L1+i
          if (s0(to)==-1) then; s0(to)=1;s0(i)=-1; end if
        else if (prob >0.75) then !down
          to=i+L1;if (to>L1*L2) to=i-(L2-1)*L1
```

```
          if (s0(to)==-1) then; s0(to)=1;s0(i)=-1; end if
        end if
      end if
    end if
  end do
  tmp=s0
end do
return
end subroutine initial
!---------------------------------------------------------------------
subroutine MonteCarlo(s,omega) !initial pattern with noise (diffusion)
integer, dimension(1:L1*L2), intent(inout) :: s
real, dimension(1:L1*L2,1:L1*L2), intent(in) :: omega
integer :: i,j,k,l
real :: prob,h
do i=1,Nsteps
  do j=1,L1*L2
    k=int(ran2(seed)*real(L1*L2))+1 !chosen neuron
    h=0.
    do l=1,L1*L2
      h=h+omega(k,l)*real(s(l))
    end do
    if (temperature==0) then !deterministic: sign(h)
      if (h<0) then; s(k)=-1; else; s(k)=1; end if
    else !Monte-Carlo
      if (exp(h/temperature) < 1.) then
        prob=exp(h/temperature)
      else
        prob=1.
      end if
      if (ran2(seed)<prob) then; l=1; else; l=-1; end if
      s(k)=l
    end if
  end do
end do
return
end subroutine MonteCarlo
!---------------------------------------------------------------------
end module fonctions
program s12ex1
use fonctions
implicit none
integer, dimension(1:L1*L2,1:M) :: sigma
integer, dimension(1:L1*L2) :: s,s0
real, dimension(1:L1*L2,1:L1*L2) :: omega
integer :: i
seed = -4856
call store(sigma)
call hebb(omega,sigma)
call initial(s0,sigma(:,chosenpattern))
s=s0
call MonteCarlo(s,omega)
open(unit=1,file=filename) !saving the data
do i=1,L1*L2 !initial pattern
```

```
  if (sigma(i,chosenpattern)==1) write(1,*) mod(i-1,L1)+1,int((i-1)/L1)+1
end do
do i=1,L1*L2 !initial pattern with noise
  if (s0(i)==1) write(1,*) (L1+1)+mod(i-1,L1)+1,int(real((i-1))/real(L1))+1
end do
do i=1,L1*L2 !pattern found
  if (s(i)==1) write(1,*) (2*L1+2)+mod(i-1,L1)+1,int(real((i-1))/real(L1))+1
end do
close(1)
end program s12ex1
```

La Fig. 1 représente un résultat de l'algorithme pour un réseau de taille 32^2 avec 2 motifs mémorisés par la règle de Hebb.

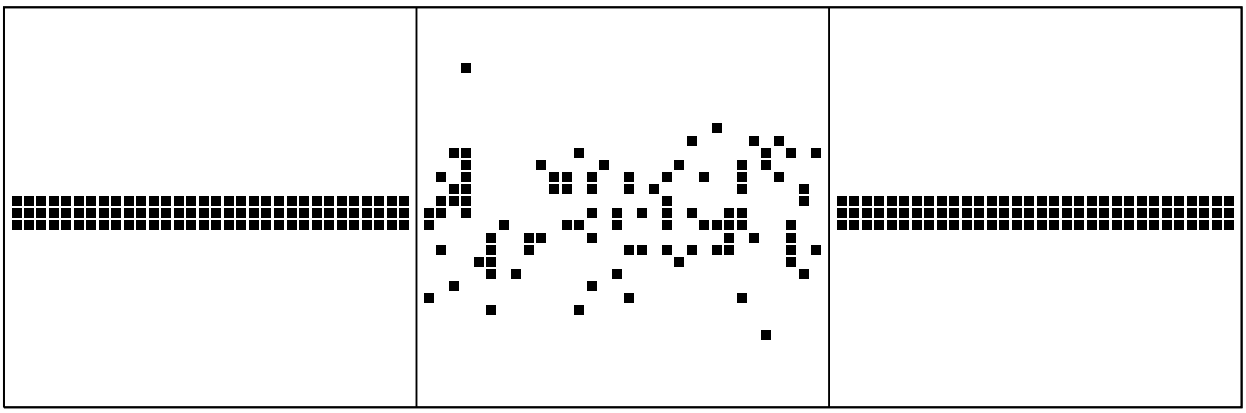

FIG. 1 – A gauche le motif à trouver, au milieu le motif bruité d'entrée, et à droite le motif trouvé par l'algorithme.

Dans ce cas, le nombre de configurations enregistrées est suffisamment petit et les motifs sont suffisamment distincts pour permettre l'identification du motif malgré un très fort bruit.