Complexity in social dynamics : from the micro to the macro
Laboratory 1 - Cellular automata

Franco Bagnoli

Namur 9/4/2008

1 Laboratory 1

Summary

1. Introduction to the laboratory.
2. FORTRAN 95 and gnuplot.
3. Cellular automata.
4. Probabilistic cellular automata (directed percolation). Trajectories and observables.
5. Phase transitions. Phase diagram.
6. The Domany-Kinzel model
7. Mean field approximation.

Introduction to the laboratory

- Language: We shall use FORTRAN 95 + gnuplot
- FORTRAN: ifort (intel), g95, gfortran (GNU)
- gnuplot (preferably versione 4.3)
- For making life easier, I suggest compiling (with a C compiler) a C wrapper for pipes (gnuplot.c, see attached material): gcc -c gnuplot.c
- For testing, try life.f90 (f95 life.f90 gnuplot.o)

Exercise 1

- Production of data with fortran90: `firstExample.f90`.
- Compilation: `f95 firstExample.f90`. Produces `a.out`.
- Redirection to a file: `./a.out > firstExample.dat`
- Plot:

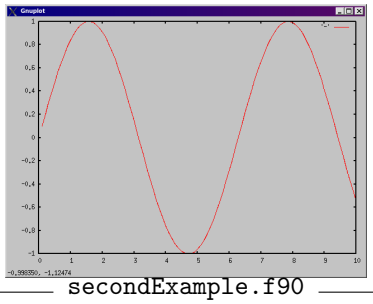
```
$ gnuplot <cr>
gnuplot> plot 'firstExample.dat' with lines <cr>
gnuplot>
```

firstExample.f90

```
program firstExample
  implicit none
  integer :: i
  integer, parameter :: N=100
  real :: x(N), y(N)

  do i=1,N
    x(i) = real(i)/10
    y(i) = sin(x(i))
  end do
  do i=1, N
    print *, x(i), y(i)
  end do
end
```

Exercise 2

- Piping data from program: `secondExample.f90`
- gnuplot can read from the same stream both commands and data.
- Just use the special filename `'-'` (and write `end` at the end).
- Fortran does not have function for dealing with pipes: use `gnuplot.c`
- Compile it with `gcc -c gnuplot.c`



`secondExample.f90`

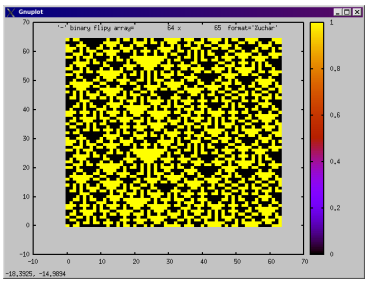
```
program secondExample
  implicit none
  integer :: i
  integer, parameter :: N=100
  real :: x(N), y(N)
  character*100, str


  do i=1,N
    x(i) = real(i)/10
    y(i) = sin(x(i))
  end do
  call gnuplotOpen("gnuplot")
  call gnuplotExecute("plot '-' with lines")
  do i=1, N
    write(str, *) x(i), y(i), char(0)
    call gnuplotExecute(str)
  end do
  call gnuplotExecute("end")
  call gnuplotFlush()
  pause
  call gnuplotClose()


end program
```

Cellular automata

- Program: `CA.f90`
- We use vector `y(0:N1)`+ for data
- first and last positions are for boundary: with periodi boundary conditions, `y(0)=y(N)` and `y(N+1)=y(1)`
- We use pointers for referring to "windows" on data
- `xl, xc, xr` corresponds to `y(0:N-1)`, `y(1:N1)`, `y(2:N+1)`
- In this way we can elaborate "in parallel" a whole configuration.

Visualization

- In the second example we sent data to gnuplot in ASCII.
- The program has to convert data from the internal representation to ASCII, and gnuplot has to parse them, and convert them back to the internal representation.
- Passing data in binary speeds up the visualization.
- In this case we have to tell gnuplot the size of data and its format.
- We use the function `gnuplotWrite` (just an interfate to `fwrite`) to send data in binary.



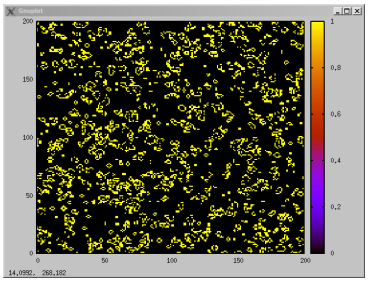
CA.f90

```
program CA
  implicit none
  integer, parameter :: N=64
  integer, parameter :: TMAX = 64
  integer*1, target :: y(0:N+1)
  integer*1 :: x1(1:N)
  integer*1, pointer :: xc(:),xl(:),xr(:)
  integer :: i, j, t
  real :: r(n)
  character*200 :: str
  integer :: rule
  integer :: tau(0:7)

  if (iargc() < 1) then
    print *, "rule number?"
    read (*,*) rule
  else
    call getarg(1, str)
    read (str,*), rule
  end if
  print *, "rule=", rule


  do i=0, 7
    tau(i) = mod(rule, 2)
    rule = rule / 2
  end do
  print *, "using rule:", tau


  xc => y(1:N)
  xl => y(0:N-1)
  xr => y(2:N+1)

  !initialization
```

```
  call random_number(r)
  xc = floor(r+0.4)   ! xc(i) = 0, 1 with prob 0.5

  call gnuplotOpen("/usr/local/bin/gnuplot")
  call gnuplotExecute("set term x11; set mouse")
  call gnuplotExecute("set cbrange [0:1]")
  write (str, *) "plot '-' binary flipy array=",N,"x",TMAX+1, " format='%uchar' " //&
    " with image", char(0)
  call gnuplotExecute(str)
  call gnuplotWrite(xc, 1, N)

  do t=1, TMAX
    !boundary conditions
    y(0)=y(N)
    y(N+1) = y(1)
    x1 = xl*4+xc*2+xr
    xc = tau(x1)
    call gnuplotWrite(xc, 1, N)
  end do

  call gnuplotFlush()
  pause
end program
```

Two-dimensional CA

- Two-dimensional CA are similar: `life.f90`
- In this case we wisualize snapshots of the configuration.
- Try to investigate the variation of the asymptotic state and the transient as a function of the initial density.



life.f90

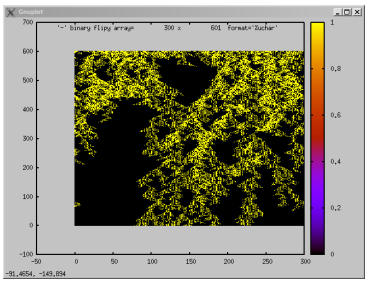
```
program life
  implicit none
  integer, parameter :: L=200
  integer*1, target :: data (0:L+1,0:L+1)
  integer*1, pointer :: C(:,:)
  integer*1, pointer :: E(:,:)
  integer*1, pointer :: S(:,:)
  integer*1, pointer :: W(:,:)
  integer*1, pointer :: N(:,:)
  integer*1, pointer :: SE(:,:)
  integer*1, pointer :: SW(:,:)
  integer*1, pointer :: NE(:,:)
  integer*1, pointer :: NW(:,:)
  integer :: i, j, t, TMAX=10000
```

```
  real :: r
  character*200 :: str

  C => data(1:L,1:L)
  E => data(1:L, 2:L+1)
  W => data(1:L, 0:L-1)
  N => data(0:L-1, 1:L)
  S => data(2:L+1, 1:L)
  NE => data(0:L-1, 2:L+1)
  SE => data(2:L+1, 2:L+1)
  NW => data(0:L-1, 0:L-1)
  SW => data(2:L+1, 0:L-1)

  call random_seed

  do i=1, L
    do j=1, L
      call random_number(r)
      data(i,j) = floor(r+0.2)
    end do
  end do

  call gnuplotOpen("gnuplot")
  call gnuplotExecute("set mouse")
  call gnuplotExecute("set cbrange [0:1]")
  write (str,*) "set xrange [0:",L,"]; set yrange [0:",L,"]", char(0)
  call gnuplotExecute(str)

  do t=1,TMAX
  ! boundary conditions
    data(:,0) = data(:,L)
    data(:,L+1) = data(:,1)
    data(0,:) = data(L,:)
    data(L+1,:) = data(1,:)
    C = 10*C+E+N+S+W+SW+SE+NW+NE
    where (C==12 .or.  C == 13 .or. C == 3)
      C = 1
    elsewhere
      C = 0
    endwhere

    write (str, *) "plot '-' "// &
    " binary array=",L,"x",L," format='%uchar'"  // &
    " title 't=",t,"'  with image ", char(0)
    call gnuplotExecute(str)
    call gnuplotWrite(C, 1,L*L)
    call gnuplotFlush()

  end do
  call gnuplotClose()
end program
```

Probabilistic CA

- The simulation of probabilistic CA is similar: `DP.f90`

- In this case the input is the probability p of percolation.
- We have to extract random numbers and compare them with p.
- Varying p we observe different scenarios. Measure the asymptotic density d.



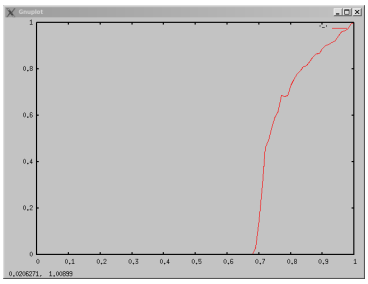
DP.f90

```
program DP !Directed percolation
  implicit none
  integer, parameter :: N=300
  integer, parameter :: TMAX = 600
  integer*1, target :: y(0:N+1)
  integer*1 :: x1(1:N)
  integer*1, pointer :: xc(:),xl(:),xr(:)
  integer :: i, j, t
  real :: r(n), p
  real :: d
  character*200 :: str

  if (iargc() < 1) then
    print *, "probability?"
    read (*,*) p
  else
    call getarg(1, str)
    read (str,*), p
  end if
  print *, "p=", p

  xc => y(1:N)
  xl => y(0:N-1)
  xr => y(2:N+1)

  !initialization
  call random_seed()
  call random_number(r)
  xc = floor(r+0.5)

  call gnuplotOpen("gnuplot")
  call gnuplotExecute("set term x11; set mouse")
  call gnuplotExecute("set cbrange [0:1]")
  write (str, *) "plot '-' binary flipy array=",N,"x",TMAX+1, " format='%uchar' " //&
    " with image", char(0)
  call gnuplotExecute(str)
  call gnuplotWrite(xc, 1, N)

  do t=1, TMAX
    !boundary conditions
    y(0)=y(N)
```

```
    y(N+1) = y(1)
    call random_number(r)
    where (xr + xl >= 1 .and. r < p)
      x1=1
    elsewhere
      x1=0
    endwhere
    xc = x1
    call gnuplotWrite(xc, 1, N)
  end do

  call gnuplotFlush()
  d=0
  do i=1, N
    d = d + xc(i)
  end do
  d = d /N

  write(*, *) "d=", d
  pause
end program
```

Phase portrait

- We want to plot the asymptotic density d as a function of p:`DPphase.f90`
- We have just to iterate the previous computations over difefrent values of p. We get a "phase portrait".
- Study what happens varying the system size and time length. What about transients before entering the absorbing state?



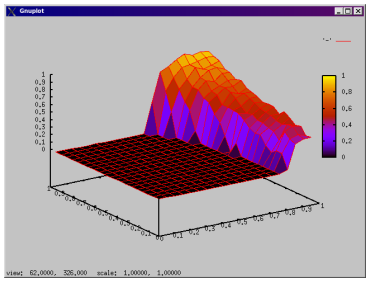
DPphase.f90

```
program DPPhase
  implicit none
  integer, parameter :: N=2000
  integer, parameter :: TMAX = 1000
  integer*1, target :: y(0:N+1)
  integer*1:: x1(1:N)
  integer*1, pointer :: xc(:),xl(:),xr(:)
  integer :: i, j, t
  real :: r(n), p=0.7
  real :: d
  character*200 :: str
  real :: pstart=0, pend=1, pstep=0.01

  xc => y(1:N)
  xl => y(0:N-1)
```

```
  xr => y(2:N+1)
  call random_seed()

  call gnuplotOpen("gnuplot")
  call gnuplotExecute("set term x11; set mouse")
  write (str, *) "plot '-' w lines", char(0)
  call gnuplotExecute(str)

  p = pstart
  do while (p < pend)
    !initialization
    call random_number(r)
    xc = floor(r+0.5)
    do t=1, TMAX
      !boundary conditions
      y(0)=y(N)
      y(N+1) = y(1)
      call random_number(r)
      where (xr + xl >= 1 .and. r < p)
        x1=1
      elsewhere
        x1=0
      endwhere
      xc = x1
    end do
    d=0
    do i=1, N
      d = d + xc(i)
    end do
    d = d /N

    write(str, *) p, d, char(0)
    print *, p, d
    call gnuplotExecute(str)
    p = p + pstep
  end do

  call gnuplotExecute("end")
  call gnuplotFlush()
  pause
end program
```

More parameters

- We can have more than one parameter. The Domany-Kinzel (DK) `DK.f90` model has two control parameters.
- Calling $p(1|A)$ the probability of getting 1 if the sum of neighbors is A, we have : $p(1|0) = 0$, $p(1|1) = p$ and $p(1|2) = q$.
- The directed percolation problem is a particular case of DK, when $p = q$ (bisectrix)
- The phase space is two-dimensional `DKphase.f90`



DK.f90

```
program DK ! Domany-Kinzel
  implicit none
  integer, parameter :: N=128
  integer, parameter :: TMAX = 200
  integer*1, target :: y(0:N+1)
  integer*1 :: x1(1:N)
  integer*1, pointer :: xc(:),xl(:),xr(:)
  integer :: i, j, t
  real :: r(n), p, q
  real :: d
  character*200 :: str

  if (iargc() < 1) then
    print *, "probability p?"
    read (*,*) p
    print *, "probability q?"
    read (*,*) q
  else if (iargc() < 2) then
    print *, "probability q?"
    read (*,*) q
  else
    call getarg(1, str)
    read (str,*), p
    call getarg(2, str)
    read (str,*), q
  end if
  print *, "p=", p, " q=", q

  xc => y(1:N)
  xl => y(0:N-1)
  xr => y(2:N+1)

  !initialization
  call random_number(r)
  xc = floor(r+0.5)

  call gnuplotOpen("gnuplot")
  call gnuplotExecute("set term x11; set mouse")
  call gnuplotExecute("set cbrange [0:1]")
  write (str, *) "plot '-' binary flipy array=",N,"x",TMAX+1, " format='%uchar' " //&
    " with image", char(0)
  call gnuplotExecute(str)
  call gnuplotWrite(xc, 1, N)

  do t=1, TMAX
    !boundary conditions
```

```
    y(0)=y(N)
    y(N+1) = y(N)
    call random_number(r)
    where ((xr + xl == 1 .and. r < p) .or. (xr + xl == 2 .and. r < q))
      x1=1
    elsewhere
      x1=0
    endwhere
    xc = x1
    call gnuplotWrite(xc, 1, N)
  end do

  call gnuplotFlush()

  d=0
  do i=1, N
    d = d + xc(i)
  end do
  d = d /N

  write(*, *) "d=", d


  pause
end program
```

DKphase.f90

```
program DKPhase
  implicit none
  integer, parameter :: N=200
  integer, parameter :: TMAX = 1000
  integer*1, target :: y(0:N+1)
  integer*1:: x1(1:N)
  integer*1, pointer :: xc(:),xl(:),xr(:)
  integer :: i, j, t
  real :: r(n), p, q
  real, allocatable :: d(:,:)
  character*200 :: str
  real :: pmin=0, pmax=1, pstep=0.05
  real :: qmin=0, qmax=1, qstep=0.05
  integer :: np, nq , ip, iq


  xc => y(1:N)
  xl => y(0:N-1)
  xr => y(2:N+1)
  call random_seed()

  np = floor((pmax-pmin)/pstep)
  nq = floor((qmax-qmin)/qstep)

  allocate(d(0:np, 0:nq))

  call gnuplotOpen("gnuplot")
  call gnuplotExecute("set term x11; set pm3d; set mouse")

  write (str, *) "splot '-' w l  ", char(0)
```

```
  call gnuplotExecute(str)


  do ip=0,np
    do iq=0,nq
      p=pmin+ip*pstep
      q=qmin+iq*qstep
      !initialization
      call random_number(r)
      xc = floor(r+0.5)
      do t=1, TMAX
        !boundary conditions
        y(0)=y(N)
        y(N+1) = y(1)
        call random_number(r)
        where ((xr + xl == 1 .and. r < p) &
          .or. (xr + xl == 2 .and. r < q))
          x1 = 1
        elsewhere
          x1 = 0
        endwhere
        xc = x1
      end do
      d(ip,iq) = 0
      do i=1, N
        d(ip,iq) = d(ip,iq) + xc(i)
      end do
      d(ip,iq) = d(ip,iq) /N
      print *,  p, q, d(ip,iq)
      write (str, *) p,q,d(ip,iq), char(0)
      call gnuplotExecute(str)
      q = q + qstep
    end do
      call gnuplotExecute("") ! gnuplot wants an empty line
                              ! after each row
      p = p + pstep
  end do
  call gnuplotExecute("end")
  call gnuplotFlush()
  pause
end program
```