Midterm

Computational Physics (**Phys381**)
R. Ouyed
February, 25, 2014
Duration: 2 Hours

[Total: 100 marks including 20 marks for the report]

- The latexed report is worth **20%** of the total mark. Only well documented and neatly presented reports will be worth that much! It is not sufficient to give only numerical results and show plots, you should also discuss your results. Include complete figure captions, introduction and conclusion sections.
- Your report must be in a *two-column format.*
- Your report should include your *Fortran code* and *Gnuplot scripts* in an Appendix using the *verbatim* command.
- You must name your report using your first and last name: *firstnamelastname.pdf.*
- **Procedure for Handing in your exam**:

  1) Set permission to your PDF report as 644. It means:
  chmod 644 *firstnamelastname.pdf*
  2) cp -a *firstnamelastname.pdf* /home/ambrish/phys381/midterm/
  3) Copy a second time to ensure that your exam copied correctly. If you are prompted as to whether or not you would like to replace the existing file, then you exam has been successfully submitted.

- **You must check with your TAs (Ambrish or Zach) that your report was received and is readable BEFORE you leave the lab.**

Attempt all questions below.

GOOD LUCK.

1 Phase Transition: A simple model

This problem makes use of the code you developed in problem # 1 in assignment # 1. You will need to extend that code and make it suitable to the problem you are asked to tackle in this exam. It also requires the use of the random number generator as described in Appendix A in this exam.

1.1 The grid

Consider a 2-dimensional lattice of squares (**a grid**), in which every square can be in one of two states: "occupied" (assigned the integer "1") or "empty" (assigned the integer "0"). Let's call the probability that a square is occupied as $p_{\text{trans.}}$ (called the **transmission probability** with $0 \leq p_{\text{trans.}} \leq 1$). To create such a grid one generates a random number r between 0 and 1 ($0 \leq r \leq 1$) for each square (i.e. site). If the random number is smaller than a chosen probability $p_{\text{trans.}}$, the square is assigned the number "1", otherwise it is assigned the number "0". The algorithm can be summarized as follows:

```
set  mygrid size: N    !!* name the grid "mygrid" *!!
set  transmission probability: p_trans
for each square [mygrid(i,j)]   get a  number  "r = random number in [0,1]"
if (r < p_trans) set mygrid(i,j)] =1  else set mygrid(i,j)=0;
end for
```

Once the grid is generated one often notices "clumps" of "1s". This is called a **cluster** of connecting squares (all assigned the number "1"). In such a clump, two squares are considered connected only if they share sides! In other words, information cannot be diagonally transmitted from one square to another.

A **spanning** cluster is a cluster of "1s" that extends across the grid. When a spanning cluster is found this means that there is an open path (a succession of connected squares with "1s") from any any side of the grid to the opposite side of the grid.

1.2 The Fortran code [50 Marks]

You will be building your code step by step as described below. However, you only need to append the completed code as an appendix in your exam. You should also append relevant gnuplot scripts you developed during this exam.

- We only consider square ($N \times N$) grids in this exam; i.e. the grid is a 2-dimensional array containing a total of N^2 squares. **Name the grid "mygrid" and declare it as an ALLOCATABLE array**. Write a Fortran 90 code that implements the algorithm above. Your code should contain two internal subroutines:

(i) *[10 Marks]* A first subroutine that generates (i.e. outputs) the **grid** for a given N and $p_{\text{trans.}}$; *name the subroutine "buildgrid"*. The random number generator should be called from inside this subroutine.

(ii) *[10 Marks]* A second subroutine that saves a generated grid into a data file to be read by gnuplot; *name the subroutine "savegrid"*. This subroutine should be called for specific values of $p_{\text{trans.}}$ (see below).

- In this first part, run your code for $N = 20$ and a transmission probability $p_{\text{trans.}}$ ranging from 0 to 1 in steps of 0.01 (i.e. $0 \leq p_{\text{trans.}} \leq 1$ with $\Delta p_{\text{trans.}} = 0.01$). It means loop over $p_{\text{trans.}}$.

  (i) *[10 Marks]* For $p_{\text{trans.}} = 0.0, 0.2, 0.4, 0.6, 0.8, 1.0$ ONLY, save your grid into a single file; *name the file "savedgrids.txt"*. **Each grid saved as a separate block in the file**.

  (ii) *[15 Marks]* Using gnuplot in a 2×3 multiplot setting, and the data in *"savedgrids.txt"*, plot the corresponding 2D color maps with the "1s" shown in color RED and the "0s" shown in BLACK (see problem #1 in assignment #1 for some guidance). The color maps should be shown clockwise with the $p_{\text{trans.}} = 0.0$ appearing in the lower left corner. Save your figure as a PDF file and add it to you report.
  Name your figure: 'yourlastname-figure1.pdf.

  (iii) *[5 Marks]* Give a brief discussion of your results.

1.3 Connectedness and spanning clusters: *The phase transition* [30 Marks]

In **Appendix B** below is a subroutine called *phasetransitioncheck* that lets you check whether the grid you generated (of size $N \times N$) contains a spanning cluster or not. It can be called from your main code as follows:

```
gridin=mygrid   !! * change the name of the grid before you pass it  *!!
Call phasetransitioncheck(gridin,N,N,result)
```

- *[10 Marks]* **From the pjl website**, copy *phasetransitioncheck* into your code and add it as an **external subroutine** to your main code; i.e. put it at the end of your code. Call it from your own code and test the output for the cases displayed in the figure you generated above (**yourlastname-figure1.pdf** for $N = 20$ and $p_{\text{trans.}} = 0.0, 0.2, 0.4, 0.6, 0.8, 1.0$). Explain what is output by *phasetransitioncheck* and in particular if it agrees with the corresponding panels in your figure.

- *[5 Marks]* Now you are asked to add an N loop to your code above; N varying from 10 to 150 in steps of $\Delta N = 10$ (as before, for each N, $p_{\text{trans.}}$ loops from 0 to 1 in steps of 0.01). Run your code and save your results into a file named *transitiondata.txt*.

This data file should have "$p_{\text{trans.}}$" as the first column and "*result*" as the second column for each N. For each N the data should be saved **as a separate block** in *transitiondata.txt*.

- *[10 Marks]* Write a gnuplot script that plots *result* versus $p_{\text{trans.}}$ for $N = 10, 50, 100, 150$ (no multi-plot is required here). **Name your figure: 'yourlastname-figure2.pdf**. Discuss the figure and its meaning in the context of spanning clusters and the connection to the concept of phase transition. In particular what behavior do you notice as N (i.e. the grid size) increases?
- *[5 Marks]* In real physical situations, phase transitions tend to occur when $p_{\text{trans.}} \sim 0.6$. Do you think the simple model you developed here reproduces that result? Can you think of a real life example where the phase transition as described in this exam can occur?

A The Random number generator

Here we illustrate the use of the random generator intrinsic to Fortran 90, **random_number(x)**. Since the computer is an entirely deterministic machine, we have to use some sort of algorithm to approximate random numbers. This is the reason why **random_seed()** must be called [but ONLY ONCE] before using **random_number(x)**.

```
!  This code generates 5 random numbers
!
!  Each random number is between 0.0 and 1.0
!
program ouyed
implicit none
integer :: i
real :: x
call random_seed()
do i = 1,5
call random_number(x)
end do
end program ouyed
```

B Check for spanning: *the routines*

```
subroutine phasetransitioncheck(grid,m,n,outcome)
  implicit none
  integer :: i
  integer, intent(in) :: n, m
  integer, intent(inout), dimension(m,n) :: grid
  integer, intent(out) :: outcome
  logical :: success, q
        outcome = 0
        success = .false.
        do i=1, m
           q = ScanMatrix(grid, i)
           success = success .or. q
        end do
        outcome = outcome + merge(1, 0, success)
contains
  logical function ScanMatrix(grid, start)
    integer, dimension(m,n), intent(inout) :: grid
    integer, intent(in) :: start
    ScanMatrix = CheckSpanning(grid, 1, start, int(start+1,1))
  end function ScanMatrix
  recursive function CheckSpanning(grid, i, j, k) result(through)
    logical :: through
    integer, dimension(m,n), intent(inout) :: grid
    integer, intent(in) :: i, j
    integer(kind=1), intent(in) :: k
    logical, dimension(4) :: q
    through = .false.
    if (i < 1) return
    if (m < i) return
    if (j < 1) return
    if (n < j) return
    if (1_1 /= grid(i, j)) return
    grid(i, j) = k
    q(1) = CheckSpanning(grid,i+0,j+1,k)
    q(2) = CheckSpanning(grid,i+0,j-1,k)
    q(3) = CheckSpanning(grid,i+1,j+0,k)
    q(4) = CheckSpanning(grid,i-1,j+0,k)
    through = (i == m) .or. any(q)
  end function CheckSpanning
end subroutine phasetransitioncheck
```