Project 3

FYS4460 - Disordered systems and percolation

Anders Johansson
19th April 2018

a)

As my Easter procrastination was reading a Fortran book[1], I decided to implement this project in Fortran. Since there is no scipy.ndimage.measurements.label in Fortran, the most important step was to implement an algorithm for labelling clusters in a binary matrix. Many high-perfomance algorithms exist for this task, and implementations of several of these in Fortran can be found on the internet. I, however, came up with a low-performance algorithm myself, and chose to implement it for fun.

My algorithm goes through the entire grid, and for each site checks if it is occupied and unlabelled. If both these conditions are met, a recursive algorithm grows the entire cluster connected to this site.

The main algorithm is the `label` subroutine,

```
subroutine label(matrix, labelled_matrix, number_of_labels)
    logical, dimension(:,:), allocatable, intent(in) :: matrix
    integer, dimension(:,:), allocatable, intent(inout) :: labelled_matrix
    integer, intent(inout) :: number_of_labels
    integer :: L, i, j

    L = size(matrix,1)
    number_of_labels = 0

    if(.not. allocated(labelled_matrix)) then
        allocate(labelled_matrix(L,L))
    end if

    labelled_matrix = 0

    do j=1,L
        do i=1,L
            if(matrix(i,j) .and. labelled_matrix(i,j) == 0) then
                number_of_labels = number_of_labels + 1
                call growcluster(matrix,labelled_matrix,i,j,number_of_labels)
            endif
        enddo
    enddo
end subroutine
```

where the recursive `growcluster` subroutine simply checks if each neighbouring site is occupied and unlabelled,

```
recursive subroutine growcluster(matrix, labelled_matrix, i, j, label)
    logical, dimension(:,:), allocatable, intent(in) :: matrix
    integer, dimension(:,:), allocatable, intent(inout) :: labelled_matrix
    integer, intent(in) :: i, j, label
    integer :: L

    L = size(matrix,1)

    labelled_matrix(i,j) = label

    if(i<L .and. matrix(i+1,j) .and. labelled_matrix(i+1,j)==0) then
        call growcluster(matrix, labelled_matrix, i+1, j, label)
```

```
            endif
            if(j<L .and. matrix(i,j+1) .and. labelled_matrix(i,j+1)==0) then
                call growcluster(matrix, labelled_matrix, i, j+1, label)
            endif
            if(i>1 .and. matrix(i-1,j) .and. labelled_matrix(i-1,j)==0) then
                call growcluster(matrix, labelled_matrix, i-1, j, label)
            endif
            if(j>1 .and. matrix(i,j-1) .and. labelled_matrix(i,j-1)==0) then
                call growcluster(matrix, labelled_matrix, i, j-1, label)
            endif
        end subroutine
```

This algorithm is easily extensible to other geometries and connectivities, as the basic idea "check all sites, check all neighbours" is generally valid. Higher dimensions should also be unproblematic, although a generalisation of checking all neighbours would be useful to avoid tedious coding.

Spanning cluster detection is done by checking if either the top and bottom rows or the left and right columns contain common elements. This is implemented by first going through one of the arrays and registering which labels are present, and then going through the other array and checking if any of the labels in this array were also present in the first,

```
            allocate(label_found(0:number_of_labels))
            label_found = .false.

            do i=1,L
                label_found(array1(i)) = .true.
            end do

            do i=1,L
                if(array2(i) /= 0 .and. label_found(array2(i))) then
                    intersect_label = array2(i)
                    return
                 end if
            end do
```

When the label of the spanning cluster has been found, the area is straightforwardly calculated by a command like `count(labelled_matrix == spanning_label)`. A sample run gives

```
Binary matrix:
   T   T   F   T   T   F
   F   F   F   F   T   T
   F   F   F   F   F   T
   F   T   F   F   F   T
   T   T   F   F   T   T
   T   T   T   T   T   T
Labelled matrix:
   1   1   0   2   2   0
   0   0   0   0   2   2
   0   0   0   0   0   2
   0   2   0   0   0   2
   2   2   0   0   2   2
   2   2   2   2   2   2
Spanning cluster: 2
Spanning cluster area:17
```

References

[1] Walter S. Brainerd. *Guide to Fortran 2008 programming*. London: Springer, 2015. 408 pp. ISBN: 978-1-4471-6758-7.

[2] L. He, Y. Chao and K. Suzuki. "A Run-Based Two-Scan Labeling Algorithm". In: *IEEE Transactions on Image Processing* 17.5 (May 2008), pp. 749–756. ISSN: 1057-7149. DOI: 10.1109/TIP.2008.919369.