

```

001 ! #####
002 ! forest fire (monte carlo percolation) on a 2D lattice simulation
003 ! stage one: percolation algorithm and the percolation threshold
004 ! john kut (jkk279@cam.ac.uk) - magdalene college - edition 2.0 - nov 2001
005 ! comments: watertight + stable edition - stage 1a completed successfully
006 ! #####
007
008 ! *****
009 module various
010 ! a module of assorted functions / subroutines and declarations
011
012 real (kind(1.0d0)) :: x, y, p_proportion, p_obtained, start_time, &
013 finish_time, processing_time
014 real :: treecount_c ! needs to be single precision!
015 integer i, j, r, forest_size, firewall, treecount, sitecount, EMPTY, TREE, &
016 BURNING, BURNT, iteration, view, lifetime, count_empty, count_tree, &
017 count_burning, count_burnt, count_empty_p, count_tree_p, count_burning_p, &
018 count_burnt_p, ix, iy, burnt_row
019 integer, allocatable :: forest(:, :), row(:), column(:)
020 character(len=1), allocatable :: gforest(:, :)
021 logical begin, nothing_left_to_burn, reached_last_row, wait
022 parameter (EMPTY=0, TREE=1, BURNING=2, BURNT=3, firewall=9)
023 ! global data type declarations, variables and parameters
024
025 contains
026 subroutine clear(n)
027 ! a basic subroutine for clearing n lines of the screen
028 ! this subroutine is frequently called in the code to insert blank
029 ! lines in the screen output for clarity - in the form of:
030 ! call clear(n) for n screen lines
031     integer n, a
032     do a = 1, n, 1
033         write (*,*)
034     enddo
035 end subroutine clear
036
037 subroutine show_forest
038 ! simply display the array on-screen:
039     print '(a9, 9999i2)', " Column: ", column(:)
040     call clear(1)
041     do i = 1, (forest_size + 2), 1
042         print 123, " Row:", row(i), " ", forest(i, :)
043     enddo
044 ! formatting statement:
045 123 format (a5, i3, a1, 9999i2)
046 end subroutine show_forest
047
048 subroutine show_forest_graphically
049 ! display the array on-screen in an easier viewing format
050 ! (i.e. not an array of numbers)
051 ! An identical array is created copying the entries of the forest array
052 ! and displaying them as suitable characters in a new (identical) array
053     do i = 1, (forest_size + 2), 1 ! running through rows
054         do j = 1, (forest_size + 2), 1 ! running through columns
055             select case (forest(i,j))
056                 ! the select case clause looks at the forest array
057                 ! of integers, and assigns them letters in a new
058                 ! text array called gforest
059                 case (EMPTY) ! an empty site
060                     gforest(i,j) = "."
061                 case (TREE) ! a tree is present
062                     gforest(i,j) = "T"
063                 case (BURNING) ! a burning tree
064                     gforest(i,j) = "b"
065                 case (BURNT) ! a burnt tree
066                     gforest(i,j) = "/"
067                 case (firewall) ! the border
068                     gforest(i,j) = "o"
069                 case default ! this should never happen
070                     call error
071             end select

```

```

072         enddo
073     enddo
074     ! Now showing the 'easy-view' 'graphical' forest:
075     print '(a9, 9999i2)', " Column: ", column(:)
076     call clear(1)
077     do i = 1, (forest_size + 2), 1
078         print 125, " Row:", row(i), " ", gforest(i,:)
079     enddo
080     ! formatting statement:
081     125 format (a5, i3, a1, 9999a2)
082 end subroutine show_forest_graphically
083
084 subroutine show
085 ! given the users preference to display the forest psuedographically or
086 ! as an array of numbers or not at all, call the appropriate subroutine
087     select case (view)
088         ! a logic variable assigned based on user input
089         case (1)
090             call show_forest_graphically
091             ! psuedographically
092         case (2)
093             call show_forest
094             ! raw number data
095         case (3)
096             ! show nothing and do nothing
097         case default
098             ! a state which should never occur
099             call error
100     end select
101 end subroutine show
102
103 subroutine forest_iteration
104 ! The iterative forest_percolation subroutine:
105 ! Passing through the forest lattice setting nearest neighbours alight
106 ! This method does not require a 2nd array - working only with the
107 ! single forest array and some prudent error checking
108
109 ! forest(2,2)=3 - A DELIBERATE ERROR TO TEST THE ERROR HANDLING
110 ! call show
111
112 nothing_left_to_burn = .true.
113 ! on each pass of this iteration routine, this variable is set to .true. -
114 ! representing that the fire HAS extinguished. As the routine scans the
115 ! forest array, it will set it to .false. if there is a tree presently
116 ! burning. There should ALWAYS be a tree burning, unless the fire has
117 ! extinguished!
118 count_empty = 0; count_tree = 0; count_burnt = 0; count_burning = 0
119 ! used for analysing the number of tree states present in each iteration
120 burnt_row = 0
121 ! used for recording how far the fire has progressed through the forest
122
123 do i = 2, (forest_size + 1), 1          ! running through the rows
124     do j = 2, (forest_size + 1), 1      ! running through the columns
125         select case (forest(i,j))      ! analyse the current cell
126             case (EMPTY)                ! an empty site
127                 ! do nothing - there is no tree there
128                 count_empty = (count_empty + 1)
129             case (TREE)                  ! a site with a tree
130                 ! a tree which is not burning, will catch fire
131                 ! if a tree to its LEFT or ABOVE is BURNT or
132                 ! a tree to its RIGHT or BELOW is BURNING
133                 ! otherwise, the tree remains unharmed
134                 if ((forest(i, j-1)==BURNT) .or. &
135                     (forest(i-1, j)==BURNT) &
136                     .or. (forest(i, j+1)==BURNING) .or. &
137                     (forest(i+1, j)==BURNING)) then
138                     forest(i, j)=BURNING
139                     count_burning = (count_burning + 1)
140                 else
141                     ! tree not able to catch fire
142                     count_tree = (count_tree + 1)

```

```

143         end if
144     case (BURNING)          ! a site with a burning tree
145         if ((i==forest_size+1) .and. &
146             (forest(forest_size+1, j)==BURNING)) &
147             reached_last_row=.true.
148             ! note if the last row has a burning tree in it
149             forest(i,j) = BURNT
150             ! a burning tree is changed to a burnt tree
151             nothing_left_to_burn = .false.
152             ! therefore, something was left in the forest
153             ! which could burn on this iteration
154             count_burnt = (count_burnt + 1)
155             !
156             if (i>burnt_row) burnt_row=i
157             ! note the furthest row down with burnt trees
158         case (BURNT)          ! a site with a burnt tree
159             ! nothing need be done, except some (optional)
160             ! error checking: there CANNOT be a nearest
161             ! neighbour tree which is not burning/burnt
162             if ((forest(i, j-1)==TREE) .or. &
163                 (forest(i, j+1)==TREE) .or. &
164                 (forest(i-1, j)==TREE) .or. &
165                 (forest(i+1, j)==TREE)) call error
166             count_burnt = (count_burnt + 1)
167             !
168             if (i>burnt_row) burnt_row=i
169             ! note the furthest row down with burnt trees
170         case (4:)          ! this case should never arise
171             call error ! error if it does
172             ! numbers 4 to 8 have no designation, and the
173             ! array is not coded to run into the
174             ! firewall of 9's
175     end select
176 enddo
177 enddo
178 end subroutine forest_iteration
179
180 subroutine prompt
181 ! wait for the user to press enter before continuing
182     print *, "Press <ENTER> to continue..."; read (*,*)
183     call clear(1)
184 end subroutine prompt
185
186 subroutine error
187 ! a subroutine to inform the user of a fatal error arising from a
188 ! supposedly impossible state
189     call clear(2)
190     print *, "A fatal error has occurred....program terminating"
191     call clear(1); stop
192 end subroutine error
193
194 end module various
195 ! end of the module
196 ! *****
197 !
198 ! *****
199 program monte_carlo_percolation
200
201 use various; use nag_f77_g_chapter, pseudorandomnumber => g05caf, &
202 nonrepeat => g05ccf; use nag_f77_x_chapter, cputime => x05baf
203 ! module usage with renaming of NAG functions to friendlier names
204 implicit none
205 call clear(60); call nonrepeat()
206 ! clear the screen by 60 lines and ensure the NAG random number
207 ! generator does not repeat
208
209 print *, "*****"
210 print *, "2d percolation lattice simulation...starting up"
211 print *, "*****"
212 call clear(3)
213 ! application startup routines

```

```

214
215 print *, &
216 "Please enter the proportion of sites (p) with trees present (between 0 to 1):"
217 10 read (*,*) p_proportion
218 ! ask the user for the value of p
219 If ((p_proportion>1) .or. (p_proportion<0)) then
220     print *, "ERROR: Please enter a value of p between 0 and 1"
221     goto 10
222 End if
223 !
224 call clear(1)
225 print *, "The value of (p) you entered was processed by the system as:"
226 print 400, p_proportion
227 400 format (f8.6)
228 ! inform the user what the system thinks of the value of p entered
229 call clear(1)
230 !
231 print *, "How do you wish to view the forest? [1,2 or 3]"
232 call clear(1)
233 print *, " 1) Pseudo-graphically"
234 print *, " 2) Raw [as numbers]"
235 print *, " 3) No Output [suitable for large forests]"
236 16 read (*,*) view
237 ! ask the user for their preference
238 If ((view .ne. 1) .and. (view .ne. 2) .and. (view .ne. 3)) then
239     print *, "ERROR: Please select an option [1,2 or 3]"
240     goto 16
241 End if
242 ! prompt the use if they want to always see the forest as an array of numbers
243 ! or pseudo-graphically (which is easier to inspect visually) or not at all
244 call clear(1)
245 !
246 select case (view)
247 ! based on the forest array viewing preference chosen previously, prompt the
248 ! user for the size of the square forest array (where the forest is limited
249 ! to 505x505 if the output is going to be printed on the screen, due to the
250 ! limitations of the "print" command
251     case (1,2)
252         ! pseudographic (1) or raw screen (2) output selected
253         print &
254         *, "Please enter the integer size of the square forest", &
255         " (1 to 505):"
256         ! prompt the user for an input
257         13 read (*,*) forest_size
258         If ((forest_size>505) .or. (forest_size<1)) then
259             print *, &
260             "ERROR: Please enter a value between 1 and 505"
261             goto 13
262         End if
263     case (3)
264         ! no screen output selected
265         print &
266         *, "Please enter the integer size of the square forest (>1):"
267         ! prompt the user for an input
268         18 read (*,*) forest_size
269         If (forest_size<1) then
270             print *, &
271             "ERROR: Please enter a value greater than 0"
272             goto 18
273         End if
274     case default
275         ! a state which should never occur
276         call error
277 end select
278 !
279 call clear(1)
280 sitecount=(forest_size**2)
281 print *, "The size of the square forest is:", forest_size, "x", &
282 forest_size, "=", sitecount, "forest sites."
283 ! state the size of the forest and the number of sites in total
284 call clear(1)

```

```

285
286 allocate (forest(forest_size+2, forest_size+2))
287 allocate (gforest(forest_size+2, forest_size+2))
288 forest=EMPTY
289 ! create the forest array of the size, entered by the user, but with a
290 ! BORDER and allocate all entries to be zero by default
291 allocate (row(forest_size+2)); allocate (column(forest_size+2))
292 ! create an array for the column and row numbers
293
294 ! the forest firewall border is characterised by the number 9
295 forest(1,:)=firewall; forest(forest_size+2,:)=firewall
296 forest(:,1)=firewall; forest(:,forest_size+2)=firewall
297 ! create the forest firewall border -- allowing subsequent array operations
298 ! to read off the edge of the forest without a problem
299
300
301 ! CREATING A FOREST OF RANDOM TREES (ORIGINAL - NOW REDUNDANT):
302 ! treecount=0
303 ! do i = 2, (forest_size+1), 1 ! running through the rows
304 !   do j = 2, (forest_size+1), 1 ! running through the columns
305 !     x = pseudorandomnumber(x)
306 !     ! call the non-repeating random number generator
307 !     if (x<p_proportion) then
308 !       forest(i,j)=TREE
309 !       treecount=treecount+1
310 !       ! assigning a site to contain a tree
311 !     elseif (x>p_proportion) then
312 !       forest(i,j)=EMPTY
313 !       ! assigning a site to remain empty (somewhat unnecessary)
314 !     end if
315 !   enddo
316 ! enddo
317 ! END OF CREATING A FOREST OF RANDOM TREES
318
319
320 ! IMPROVED CREATION OF A FOREST OF RANDOM TREES (NEW): ****
321 treecount = anint(real(sitecount) * p_proportion)
322 ! given the input value of p, the system needs to distribute 'treecount'
323 ! number of trees into the forest, randomly. If sitecount * p_proportion is
324 ! not already an integer, it will be rounded to one
325 treecount_c = (real(sitecount) * p_proportion)
326 ! do the same calculation as above, but without rounding and keeping it real
327
328 if (treecount .ne. treecount_c) then
329 ! If p entered is not valid for the size of the forest chosen, warn the user
330 print *, &
331     "** CAUTION: The size of the forest and the value of p entered does not"
332 print *, "result in an integer number of trees! ", &
333     "This will be rounded accordingly."
334 call clear(1); call prompt
335 end if
336 ! if the treecount value was rounded, the p entered by the user cannot be
337 ! produced exactly (it's unphysical to have a non-integer number of trees in
338 ! a forest!)
339
340 print *, "Generating the forest....placing", treecount, &
341 "trees randomly in", sitecount, "spaces"
342 ! plant trees by picking spaces in the forest at random and planting a tree
343 ! until you have planted all the trees necessary. If the site already has
344 ! a tree, pick another space randomly until you find an empty space
345
346 ! an extra (unnecessary?) level of randomness could be to pick the spaces at
347 ! random and then choose randomly whether or not to plant a tree there if
348 ! it is an empty site - however given the apparent and analysed randomness
349 ! of the existing approach - this would appear to be satisfactory as it is
350 do r = 1, treecount, 1
351 ! run through the number of trees which needs to be planted in
352 ! the forest and plant them randomly
353 20 x = pseudorandomnumber(x); y = pseudorandomnumber(y)
354 ! label 20: call the non-repeating random number generator for
355 ! real variables x and y, where the random numbers are 0 < x < 1

```

```

356     if (x==1 .or. x==0 .or. y==0 .or. y==1.) call error
357     ! the random numbers are therefore never equal to 1 or 0 (hopefully)
358     ! error if for some strange reason they are
359     ix = (1+int((real(forest_size)) * x))
360     iy = (1+int((real(forest_size)) * y))
361     ! the random numbers x and y are converted into random positions in
362     ! the forest array, ix and iy (integers)
363     if (forest((ix+1),(iy+1))==EMPTY) then
364         forest((ix+1),(iy+1))=TREE
365         ! print *, "planting tree no.", r, "in", (ix), ",", (iy)
366         ! if the site does not contain a tree, plant a tree there
367     elseif (forest((ix+1),(iy+1))==TREE) then
368         ! print *, "planting tree no.", r, "in", (ix), ",", (iy)
369         ! print *, "...already contains a tree...trying elsewhere"
370         goto 20
371         ! if the site already contains a planted tree, skip it and
372         ! find another random site
373     else
374         ! print *, forest((ix+1), (iy+1)), ix, iy
375         call error
376         ! this should never happen
377     end if
378 enddo
379 print *, "...done"
380 ! END OF IMPROVED CREATION OF A FOREST OF RANDOM TREES ****
381
382 do i = 1, (forest_size + 2), 1
383     row(i)=i-1; column(i)=i-1
384 enddo
385 ! fill the row and column array with their row and column numbers
386 call clear(1)
387 print *, &
388 "....."
389 print *, &
390 ". FOREST KEY:                ."
391 print *, &
392 ". An empty site is    0 or .    A tree is    1 or ¶ ."
393 print *, &
394 ". A burning tree is    2 or b    A burnt tree is 3 or / ."
395 print *, &
396 ". The firewall border is    9 or °                ."
397 print *, &
398 "....."
399 ! Print on-screen key information for the forest
400 call clear(1); call show
401 ! call the subroutine to display the forest on the screen
402 ! state information about the forest:
403 call clear(1)
404 p_obtained = (real(treecount)/real(sitecount))
405 print 401, "The number of trees in the forest is:      ", &
406 treecount, (p_obtained*100.), "%"
407 print 401, "The number of empty sites in the forest is: ", &
408 (sitecount-treecount), ((real(sitecount-treecount)/real(sitecount))*100.), "%"
409 print 401, "The total number of sites in the forest is: ", sitecount, &
410 ((real(sitecount)/real(sitecount))*100.), "%"
411 print 402, "The actual proportion (p) obtained is:      ", p_obtained
412 print 402, "The proportion (p) originally entered was:  ", p_proportion
413 print 402, "The absolute difference between the two is: ", &
414 abs(p_obtained-p_proportion)
415 401 format (a45, i8, f10.4, a2)
416 402 format (a45, f18.6)
417 call clear(1)
418
419 ! ----- END OF FOREST CONFIGURATION AND DISPLAY -----
420
421 ! ----- START OF TREE BURNING -----
422 print *, "Do you wish to start the first nearest neighbour percolation? [T/F]"
423 read (*,*) begin ; if (.not. begin) goto 11
424 ! user prompt to continue
425
426 start_time = cputime()

```

```

427 ! call the nag routine to note down the current cpu time for later comparison
428 call clear(10)
429 print *, "Setting alight all first row (row 1) trees..."; call clear(2)
430 !
431 do j = 2, (forest_size + 1), 1
432 ! set alight any first row (row 1) trees, by setting 1 to 2
433     select case (forest(2,j))
434         case (EMPTY)
435             ! do nothing - there is no tree there
436         case (TREE)
437             ! if there is a tree there, start it burning
438             forest (2,j) = BURNING
439         case (2:)
440             ! this case should never arise - error if it does
441             call error
442     end select
443 enddo
444
445 iteration=0
446 ! set the initial iteration count to zero
447 reached_last_row=.false.
448 ! note that the fire has not yet reached the last row in iteration 0
449 call show; call clear(1)
450 ! call the subroutine to display the forest on the screen
451 Print *, "Iteration: ", iteration; call clear(1)
452 ! print the iteration count
453
454 ! Need a preliminary analysis of the forest:
455 count_empty_p = 0; count_tree_p = 0; count_burnt_p = 0; count_burning_p = 0
456 do i = 2, (forest_size + 1), 1      ! running through the rows
457     do j = 2, (forest_size + 1), 1  ! running through the columns
458         select case (forest(i,j))  ! analyse the current cell
459             case (EMPTY)          ! an empty site
460                 count_empty_p = (count_empty_p + 1)
461             case (TREE)           ! a site with a tree
462                 count_tree_p = (count_tree_p + 1)
463             case (BURNING)       ! a site with a burning tree
464                 count_burning_p = (count_burning_p + 1)
465             case (BURNT)        ! a site with a burnt tree
466                 count_burnt_p = (count_burnt_p + 1)
467             case (4:)           ! this should never happen
468                 call error ! error if it does
469         end select
470     enddo
471 enddo
472 !
473 ! print out information on the trees from the preliminary analysis:
474 print *, "A", forest_size, "x", forest_size, "forest of", sitecount, &
475 "total site(s), of which:"
476 print 404, count_tree_p, " are unburnt trees", &
477 count_empty_p, " are empty spaces"
478 print 404, count_burning_p, " are burning trees", &
479 count_burnt_p, " are burnt trees"
480 !
481 call clear(1); print *, "-----"; call clear(1)
482 !
483 print *, "Do you wish to pause after each iteration? [T/F] "
484 ! ask the user whether or not to pause after each iteration
485 read (*,*) wait; call clear(20); goto 12
486
487 ! ----- START OF LOOPING ITERATIONS -----
488 17 call prompt
489     ! prompt the user to continue
490     call clear(20)
491     !
492 12 call forest_iteration
493     ! call the subroutine to iterate once with the spreading fire
494     If (nothing_left_to_burn) goto 14
495     ! if there is nothing left to burn in the forest, exit the loop
496     If (reached_last_row) goto 15
497     ! if the fire has reached the last row, exit the loop

```

```

498     call show
499     ! call the subroutine to display the forest on-screen
500     !
501     iteration=(iteration + 1); call clear(1)
502     ! increase the iteration counter
503     print *, "Iteration: ", iteration; call clear(1)
504     ! print the iteration count
505     ! print out information on the trees:
506     print *, "A", forest_size, "x", forest_size, "forest of", sitecount, &
507     "total site(s), of which:"
508     print 404, count_tree, " are unburnt trees", &
509     count_empty, " are empty spaces"
510     print 404, count_burning, " are burning trees", &
511     count_burnt, " are burnt trees"
512     call clear(1)
513     print *, "Row", (burnt_row - 1), &
514     "is the furthest down containing burnt trees"
515     !
516     404 format (i12, a, i12, a)
517     call clear(1); print *, "-----"; call clear(1)
518     !
519     count_empty_p = count_empty; count_tree_p = count_tree
520     count_burning_p = count_burning; count_burnt_p = count_burnt
521     ! This buffers the previous iteration state information, should this
522     ! iteration turn out to be the terminating iteration
523     select case (wait)
524     ! repeat the loop and continue or pause, based on the user preference
525     case (.true.) ;      goto 17      ! pause
526     case (.false.);    goto 12      ! don't pause
527     end select
528
529     ! ----- END OF LOOPING ITERATIONS -----
530
531     14 print *, "**** END OF SIMULATION: SUMMARY ****"; call clear(1)
532     print *, "The fire has EXTINGUISHED ITSELF after", iteration, &
533     "iteration(s)"
534     goto 19
535     !
536     15 print *, "**** END OF SIMULATION: SUMMARY ****"; call clear(1)
537     print *, "The fire has REACHED THE LAST ROW of the forest in", &
538     iteration, "iteration(s)"
539     goto 19
540     !
541     19 call clear(1)
542     lifetime=iteration
543     print *, "Fire lifetime (tau): ", lifetime, "cycle(s)"
544     print *, "Forest size:      ", forest_size, "x", forest_size, "=", &
545     sitecount, "site(s) with", treecount, "tree(s)"
546     !call clear(1)
547     print 403, "The proportion (p) of trees in the forest is: ", p_obtained
548     403 format (a47, f10.8)
549     call clear(1)
550     ! print final iteration forest state information:
551     print *, "Summary of the final iteration forest state:"
552     print 404, count_tree_p, " are unburnt trees", &
553     count_empty_p, " are empty spaces"
554     print 404, count_burning_p, " are burning trees", &
555     count_burnt_p, " are burnt trees"
556     !
557     call clear(1)
558     finish_time = cputime(); processing_time = finish_time - start_time
559     ! call the NAG routine to find out the current CPU time
560     if (processing_time<100) then
561     ! format the time taken statement
562     print 405, &
563     "The forest fire iteration(s) took a total of: ", &
564     (processing_time), "seconds"
565     else
566     print 405, &
567     "The forest fire iteration(s) took a total of: ", &
568     (processing_time/60.), "minutes"

```

```
569 end if
570 print *, "(approximately) of CPU processing time to complete"
571 ! print the time it took to perform the forest iterations in total
572 405 format (a47, f10.6, a8)
573
574 ! ----- APPLICATION SHUTDOWN -----
575 11 call clear(1)
576 print *, "*****"
577 print *, "2d percolation lattice simulation...exiting"
578 print *, "*****"
579 print *, "                ", &
580 "shutting down...goodbye"
581 !
582 deallocate (forest, column, row, gforest)
583 ! deallocate the arrays used in the program
584 end program monte_carlo_percolation
585
586 ! end of the application
587 ! *****
588
```