4. The drunken sailor problem

Introduction

For my final project in Scientific Computing II I examined two types of random walks, normal random walks (NRWs) and self-avoiding random walks (SAWs). An NRW can go backwards in its path and cross its own path, resulting in longer walks. A SAW can't visit any of the points it has previously visited, and ends if all the possible points around are already visited ones. I made a program that uses these walks to simulate the movement of a drunken sailor who is trying to make it to her ship at the shore (x=10) before the morning comes (within 10 hours). The program takes 3 inputs from the user through command line: The amount of simulated walks, whether to write paths to files (for example to draw plots), and whether the user wants an NRW or a SAW. The program then returns the average, minimum and maximum distances and times walked, and the fractions of walkers who got to the ship on time, died of old age (50 years passed without reaching the beach, NRW only) and walked into dead ends (SAW only). Finally I compare the statistics of SAWs and NRWs and visualize different cases of both walks.

Methods

The program reads the number of sailors(n), whether to use trajectories (traj='yes' or 'no), and walktype ('NRW' or 'SAW') from command line. Depending on if character walktype is 'NRW' or 'SAW', the program calls the corresponding subroutine nrw or saw. Both of the subroutines work similarly. The mtfort90.f90 file contains a custom-made random number generator which we use to take steps in random directions. The sailor program calls from mtfort90.f90 modules mtdefs and mtmod. This allows us to use a function igrnd(1,4) to generate a random number in the interval [1,4], with each integer corresponding to a direction. The seed comes from calling function sgrnd(getseed()). After calling the seed, there is a do-loop that repeats the walk process for each sailor.

The walk is executed with a do-loop in NRW. Since the sailor can walk for 50 years maximum, and each step is a minute, the maximum sailor age (or amount of steps) is agemax=50*365*24*60. A do-loop from age=1,agemax is used to create the walk. Every loop calls igrnd(1,4) for a random number ran, and takes a step a direction determined by it. The program uses a convention where ran=1 is astep in y-direction, ran=2 is a step in x-direction, ran=3 is a step in -y-direction and ran=4 is a step in -x-direction. After a step is taken, the program checks using if statements if the sailor is at the beach and if she is, it checks if she got there before the morning. Making it to the beach will add 1 to the successes counter and exit the random walk loop and go to the next sailor. The coordinates coord(1) for x and coord(2) for y after each step are added to the coordhistory array. At the end of the do-loop an if-statement checks if age is agemax, so if the looping goes until age=agemax, it'll add 1 to variable agedeaths. At the start of each sailor loop we reset the coordinates and coordinate history from the previous sailor.

The SAW is similar, but the random walk can be done using a do while-loop. This loop runs while the x-coordinate is less than 10. The first step of SAW can be taken in any direction, so this is done for each sailor using the NRW structure. The subsequent steps however need to be to coordinates which haven't previously been visited by this sailor. This means the sailor can end up in coordinates from which the surrounding 4 coordinates are already visited previously, a dead end. The way the program checks if the sailor is in a dead end is by using a dead end counter. The idea is, that when we take a step in one of the 4 possible directions, we check the new (x,y) coordinates against all the old coordinates located in the coordhistory array. If these coordinates are in the coordhistory array, we simply take a step back as if nothing happened and take another random step. Taking a step back means we use cycle to repeat the loop and the sailor age (step

counter) doesn't go up. Every time we have to take a step back, we add 1 to the deadendcounter variable. If we have to take, say, 100 (or some other large number) steps back to the same coordinate, it's extremely likely we are in a dead end. Thus the program concludes the sailor is in a dead end and adds 1 to the dead ends. This is of course a probabilistic argument, but it is very simple. The highest chance for a false positive would be when there is 1 way to go, but the sailor steps in the 3 other directions a 100 times in a row. The chances for this are $(3/4)^{100} = 3.2 * 10^{-13} \approx 0$. We can safely conclude this will not affect any of the statistics in the slightest. The deadendcounter variable is set to 0 after each successful step (the coordinate we stepped to was a new one) and every time a new sailor is considered. Since the random walk loop is a do while loop, the age gets 1 added to it at the end of the loop (so cycles upon repeated coordinates don't affect the age). Similarly the coordinates are stored in coordarray at the end of the loop. The while loop ends at x=10 or if the sailor walks into a dead end, so the condition for making it to the beach in time is checked after the do while loop ends using an if (x.gt.10.and.age.lt.600) then -statement. The coordinates are again stored in coordhistory array after every step. When the program starts on a new sailor, it also resets age in the SAW case, as it's not the walk loop variable.

For both SAWs and NRWs the writing of trajectories to files is done in the same way. First the program checks if the user wanted trajectories or not by checking if traj=='yes'. If so, A string 'n.dat' is written, where n is sailor index. The full path of the data file is then constructed as a string and opened. The whole coordhistory array is written into the file with a do loop and an implied loop. These data files can be found in the trajectories folder along with a python file that can be used to draw trajectories.

After the looping process is done for all n sailors, the relevant statistics for the walk type handled (ages/distances/event probabilities) are printed. In both walks, the number of steps taken by each sailor is stored into an array nofsteps, from which the average, min and max amounts of steps are easy to extract using the sum(), minval() and maxval() intrinsics. The amount of steps is the amount of coordinates visited-1. So if one visits two coordinates, they started in one and took one step to the second coordinate. Every step is 100m, so the distance travelled in meters is obtained by multiplying the number of steps by 100, or 0.1 to get kilometers. All the statistics are handled by module stats, which includes 2 subroutines. Subroutine time() for changing minutes to wanted units and making sure the units that are 0 are not printed. The different possible combinations of time units are handled 1 by 1, to a total of 15 cases. The 16th case is the one where all units(y,d,h,m) of time are 0, which is not possible, since our obviously lasts over 1 step. Subroutine statistics() handles the printing of the min, max and avg times

and distances and probabilities of failure and success in different cases for both NRW and SAW. It takes in 7 arguments, walktype (NRW or SAW), number of sailors(n), successes (made it to beach in time), fails(died of old age[NRW] or reached dead end[SAW]), maximum amount of steps taken by a sailor, minimum steps taken and avg steps taken. The steps taken are also the max,min and average walk times since a single step takes one minute.

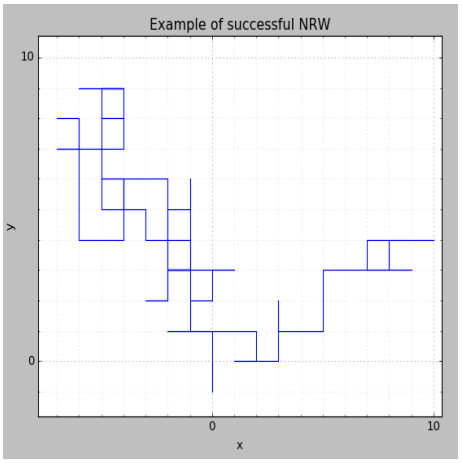
Implementation

First the main file sailor.f90 has to be linked to the file with the random number generator (mtfort90.f90) and the file with all the statistics (stat.90). With gfortran one can compile using 'gfortran -c stat.f90 mtfort.f90 sailor.f90' and link afterwards using 'gfortran sailor.o stat.o mtfort90.o' in the command line. After this, the program can be run using './a.out a b c'. The a b and c are the required command arguments for the program to run. A is the number of sailors, b is 'yes' or 'no' for trajectory files and c is 'saw' or 'nrw'. For example, 'gfortran sailor.o stat.o mtfort90.o ./a.out 10000 no saw'. Be aware that writing trajectory files for excessively large amounts of sailors can be space consuming. It isn't a problem for a few thousand sailors. The recommended amount of sailors for a fair sample size is around 10000. The expected run time for that is around 6-8 minutes (NRW), and a little less for SAW. The run time heavily depends on whether we write trajectories to a file (few times slower if we do), so for statistics purely it is highly recommended to not write trajectories to files. Opening the text files for especially the long walks is also not advised.

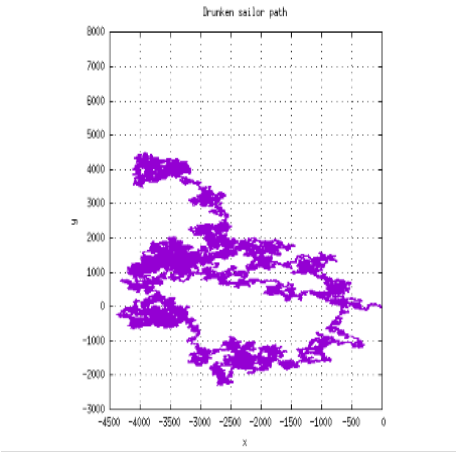
Results

Running the NRW for 100000 sailors took about 110 minutes and of all the sailors 56364 made it to the beach in time and 220 died of old age, resulting in a probability of 56.36% to make it to the ship by morning and 0.22% chance to die of old age before making it to the beach at all. The rest (43.42%) make it to the beach, but too late. The NRW minimum distance was 1.1km (11 minutes), maximum distance the expected $2.68\ 10^6$ km (50 years) and average distance about 13755 km (95 days, 12 hours and 29 minutes). Examples of a successful NRW and an NRW that lasted until death can be seen in figure 1. The deathwalk was done using gnuplot.

Running the SAW for 100000 sailors took roughly 60 minutes and of all the sailors 25054 made it to the beach in time and 74946 walked into dead



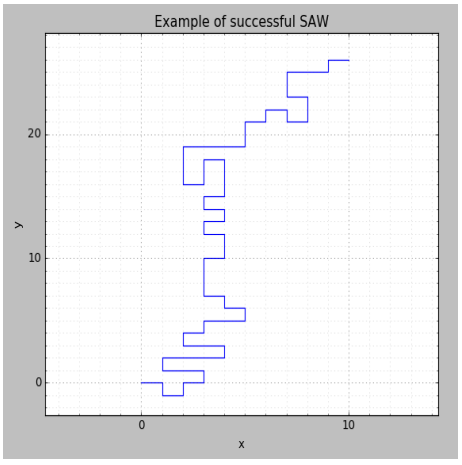
(a) Successful NRW



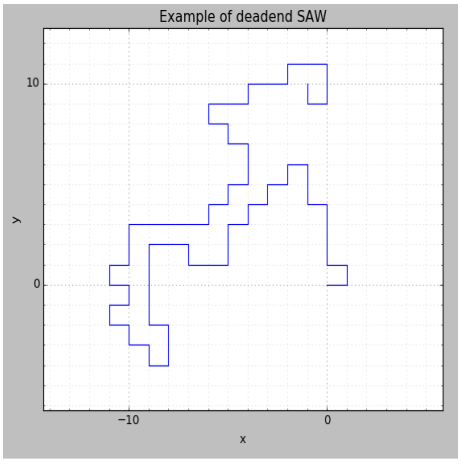
(b) NRW until death (Gnuplot)

Figure 1: The two possible NRW scenarios

ends. This gave probabilities 74.95% for walking into a dead end and 25.05% for making it to the beach in time. Not a single sailor made it to the beach late. The SAW minimum distance was 0.7km (7 minutes), maximum distance 45.8km (7 hours 38 minutes) and average distance 5.6km (56 minutes). Note that the maximum walk time was 458 minutes, much less than 600 minutes. It is thus very unlikely to have a walk both go longer than 600 minutes (10h) and make it to the beach after. Examples of a both successful SAW and a dead end scenario are illustrated in figure 2.



(a) Successful SAW



(b) Dead end SAW

Figure 2: The two possible SAW scenarios

Conclusions

The probability of successfully walking to the beach by morning in the case of NRW is 56.36%. The chance of making it, but late, is 43.42%. In 0.22% of cases the sailor dies of old age. SAW walk sees a considerable fall in chances of making it to the beach, with only 25.05% making it in time. 74.95% of the sailors got stuck in dead ends. The chance to make it to the beach, but late, is in practice 0, and no such walk was observed. The reason the SAW walk has a lower chance to make it to the beach by morning is the much lower average walk time/distance. A SAW only lasts 56 steps on average, while the average NRW lasts 137550 steps. The NRW average is of course skewed as some walks last 50 years so other statistical quantities like medians would give a more reasonable comparison. The SAW minimum distance observed, 7 steps, was the shortest possible for a SAW: A 'spiral' walk. The minimum observed length NRW was a near-straight walk to the beach, 11 steps.