Drunken sailor problem

Teemu Stepanoff

Report
Scientific computing 2
Submitted

December 18, 2020

Contents

1 Introduction

The drunken sailor problem consists of a sailor and (x, y) coordinate system where shore is set to $x = 0$. Sailor wakes up in coordinates $(10, 0)$ and starts moving in random directions wanting to get to ship which is in shore. The ship leaves in 10 hours and sailor's walkind speed is 100 m/s. The city constructs of 100m $\times$ 100m citybloks. Sailor is so drunk that she just turns to a random direction on every time until she reaches shore or dies of old age (50 years). Outcomes of this are that sailor gets to ship on time, gets to shore but misses the ship or dies of old age lost in the infinitely large city(city is infinte in $\pm$y and +x direction). Let's imagine sailor gets to ship on time and makes up a new tactic for next time. Sailor gets drunk again and now places bottles in places where she has been and avoids turning to those directions when choosing path. This movement is generally known as Self Avoiding Walk (SAW). There is an obviuos problem. The sailor can walk herself in a dead end. In this situation we assume that she gives up and drinks rest of the rum and passes out.

With these boundary conditions a program can be written to simulate the walking in both cases. This program will calculate distance walked and spent time (average, minimum, maximum) and the fractions of times sailor made to ship in time, died of old age (normal walk only) or walked in a dead end(SAW only).

2 Methods

2.1 Architecture of the program

The program costructs of four modules and one main file. The highest module is mtfort90 which functions and subroutines are inherited by all other methods. A derived type "location" is defined in module SAW. Location has three integer parameters (x,y,t), which are used to save sailors location in (x,y) and elapsed time t.

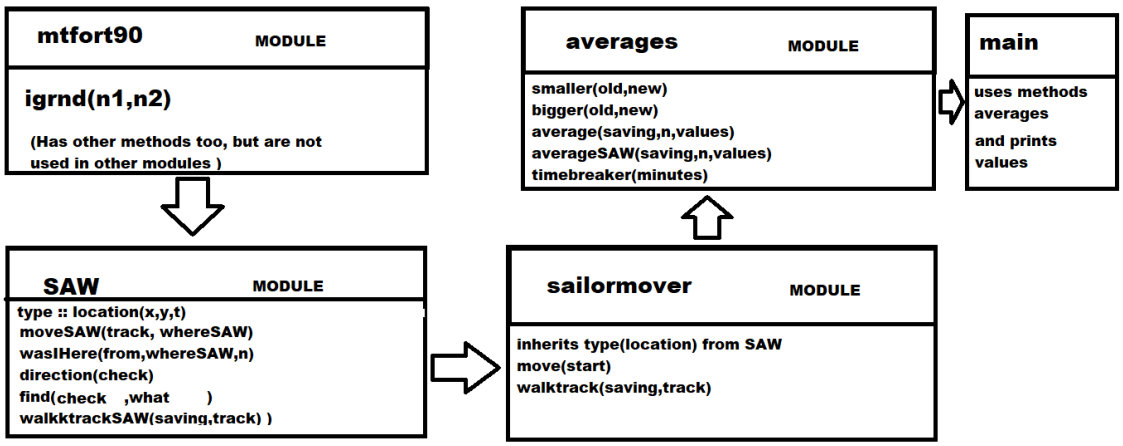


Figure 1: Arrow indicates which direction methods are inherited

2.2 mtfort90.mod

This module is directly copied from coursematerial. From here we only use method integer function igrnd(n_1, n_2) which returns random integer from region $[n_1, n_2]$. Random is depedant on modules defaultseed "defaultstd" which has to be changed explicitely from the source code to get different random numbers. To change seed for this you have to go change it inside mtfort90.f95.

2.3 SAW.mod

2.3.1 integer function find(check,what)

Parameters: "check" is integer,dimension(4) and "what" is an integer. The function gets the value of first index of which element has the value "what". If there aren't any elemnts of value "what", function gets value zero.

2.3.2 Integer function direction(check)

Returns random integer from range [1,4],depending on which way to move. Check is an integer array with dimension(4). Each element describes direction which to move. If element equals 1 the sailor has been on that location and if element equals 0 sailor hasn't been in that location before. Direction() is optimized in a way that it checks

least computing effort demanding options first. If there are two or more directions to go, igrnd() determines the direction.

2.3.3 subroutine wasIHere(from,whereSAW,n)

Returns parameter integer,dimension(4) array "n" which elements will be copied of surrounding values of "from" in "whereSAW". "from" is a derived type location, which describes coordinates where sailor is about to make the next move. "whereSAW" is our coordinate system represented as matrix (610,1200). x-axis reaches 610, beacause sailor starts in position (10,0). In unlikely scenario sailor moves only one direction so this has to be taken to account.

2.3.4 subroutine moveSAW(start,whereSAW)

This subroutine is the method that actually moves sailor's position. "start" is derived type location and whereSAW is matrix containing the information where sailor has been. Subroutine calls other subroutine wasIHere(start,whereSAW,n) and gets values for n. Then it uses value of direction(n) with now determined array "n" , to determine which direction to go (up,down,left,right) or has it come to a dead end in which case value of "start" stays the same. If sailor moves, the value of given axis is changed by ± 1 and time grows by one minute.

2.3.5 subroutine walktrackSAW(saving,track)

Subroutine uses integer parameter "saving" to determine if all location are to be saved in file "output.dat". If saving == 1, saving is done and otherwise not."track" is derived type location, which subroutine will return. It is the final location which sailor ends up. "whereSAW" matrix's values are all set to zero and starting location is set as (10,0,0).The subroutine has a loop up to 600 iterations (represents 10 hours in minutes when ship leaves). In each iteration walktrackSAW calls for subroutine moveSAW(track,whereSAW) to move the sailor. The program exits loop, if sailor has no direcions left to go, or she has made it to the beach.

2.4 sailormover.mod

2.4.1 subroutine move(start)

This is simpler version of moveSAW() method. move() changes given input parameter "start" x or y coordinate randomly negative or positive direction. Subroutine returns parameter changed value of "start".

2.4.2 subroutine walktrack(saving,track)

Saving is integer, which determines if location are saved to a file. Track is derived type location, and returnes it as the final location the sailor. Track is set (10,0,0) in every call of subroutine. Subroutine uses method move() to get new location for sailor and does this until she gets to the shore, or dies of old age.

2.5 averages.mod

2.5.1 integer function smaller(old,new)

Gets value of smaller integer (old or new). If they are the same, function gets the value of "old".

2.5.2 integer function bigger(old,new)

Works same as the method smaller(). This time function will just get the value of bigger integer.

2.5.3 chracter(len = 80) function timebreaker(minutes)

Breaks time given in minutes (integer) in to easier form to read (y years, m months, h hours, m minutes). Displays only non-zero values. The transfomed values are written into character.

2.5.4 subroutine average(saving,n,values)

Integer "saving" is passed on to subroutine walktrack($saving, track$) to determine if to save data to file or not. Integer "n" is given to the subroutine and determines how many times it calculates random paths for the sailor (using walktrack()). "Values" is subroutines input and output array. It contains the information (avergdistance, mindistance, maxdistance, avrgtime, mintime, maxtime, gotToShip %, Died %). The spent tie of one iteration is added to a variable "minutes". Spent minutes can be claimed by calling walktrack(saving,track) and then using place% t .With each every iteration loop also checks if spent time is the currently biggest or smallest spent time. Walked distance can be calculated directly from time average, minium and maxium valeus ($1min = 100meters$). "values" is a character array, so all values have to written as characters. Before that timebreaker() is used to make average, min and max times more readable.

2.5.5 averagesSAW(saving,n,values)

Functionality is same as for average() but now it calls subroutine walktrackSAW(saving,track). Only the alst element in list "values" is changed it's meaning. It now tells percentage of how many times sailor walked to a dead end. Inside loop walking to a dead end can be determined if walktrack() returns location which time ¡= 600 minutes (10 hours) and x coordinate is not zero.

3 Implementation of methods

Implementation of all module methods are already described excpet for average() and averageSAW().

As shown in figure 2.1 main only uses methods averages() and averagesSAW(). Main gets parameters for these as comand line arguments (./a.out iterations save? SAW?). "iterations" has to be an integer, otherwise program won't run. To save trajectories in to a file "save?" must be "yes" and same for "SAW?" if you want SAW simulation. Anything else in 2nd and 3rd command line arguments and program will not save/run as SAW. If $SAW? =$ "yes" progrmam will only run SAW simulation and vice versa. If parameters are absent, progrmam won't simulate anything. Program will print out how many iterations was done, by which method and did we save to a file. Program also prints the above mentioned time and distance valued for selected walking style.

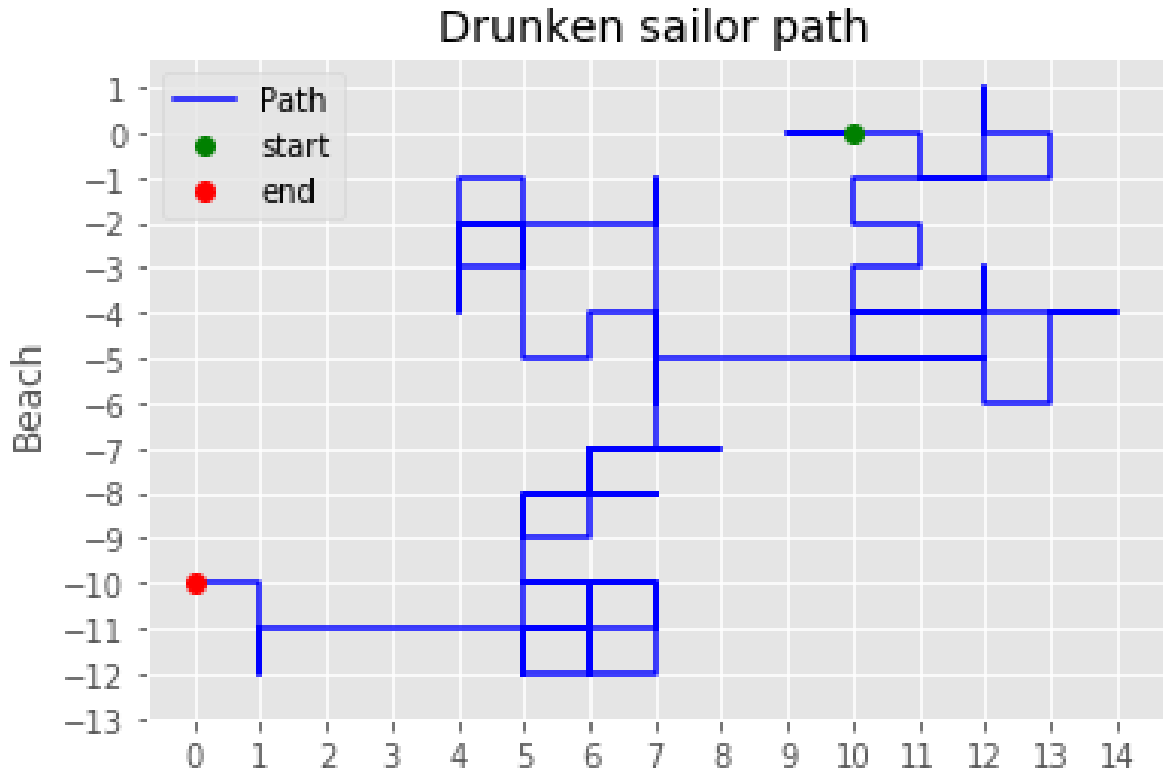


Figure 2: Normal walking gets sailor to the ship in time

4 Results

Simulations = 10 000	Ordinary walk	SAW
Average time	83 days 19hours 14 mins	54 minutes
Minium time	16 mins	7 minutes
Maxium time	50 years	7 hours, 2 minutes
Average distance	12067.4km	5.4 km
Minium distance	1.6km	.7 km
Maxium distance	2 628 000km	42.2 km
Got on ship	56.5 %	36.1 %
Died or dead end (SAW)	0.3 %	63.9 %

The trajectories of ordinary walk and can be plotted by saving data to file.

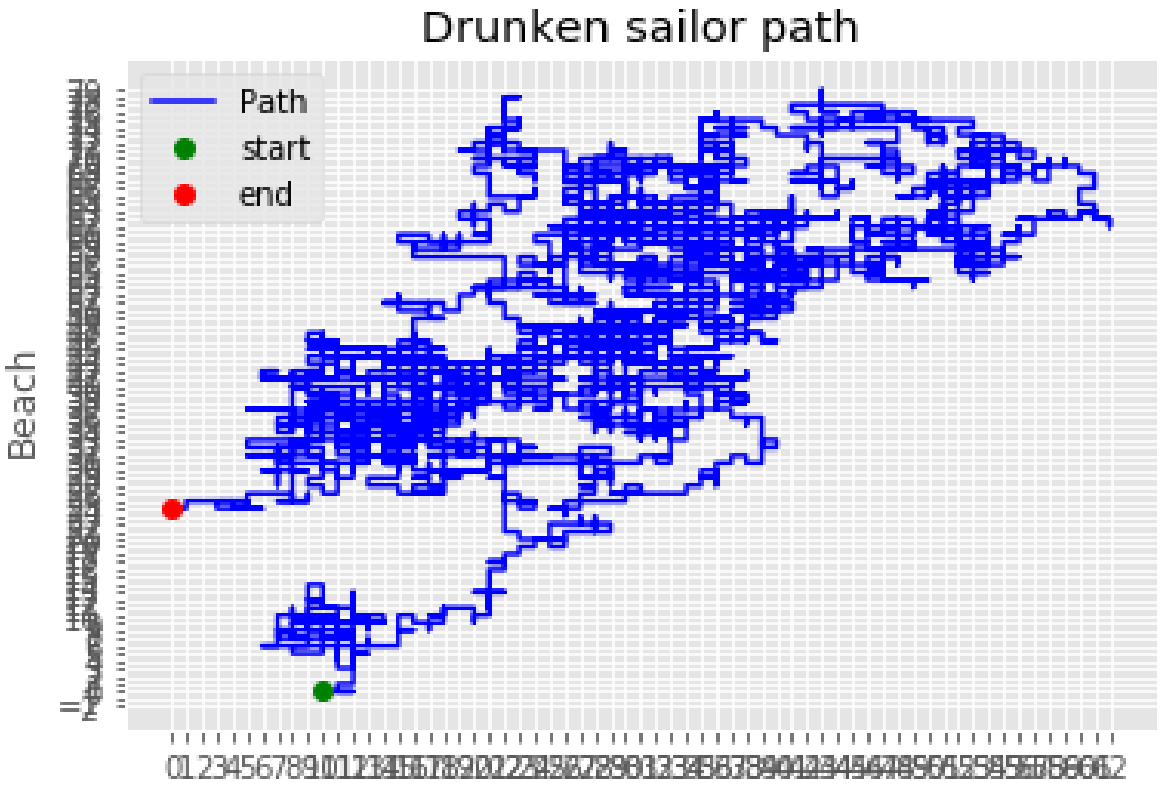
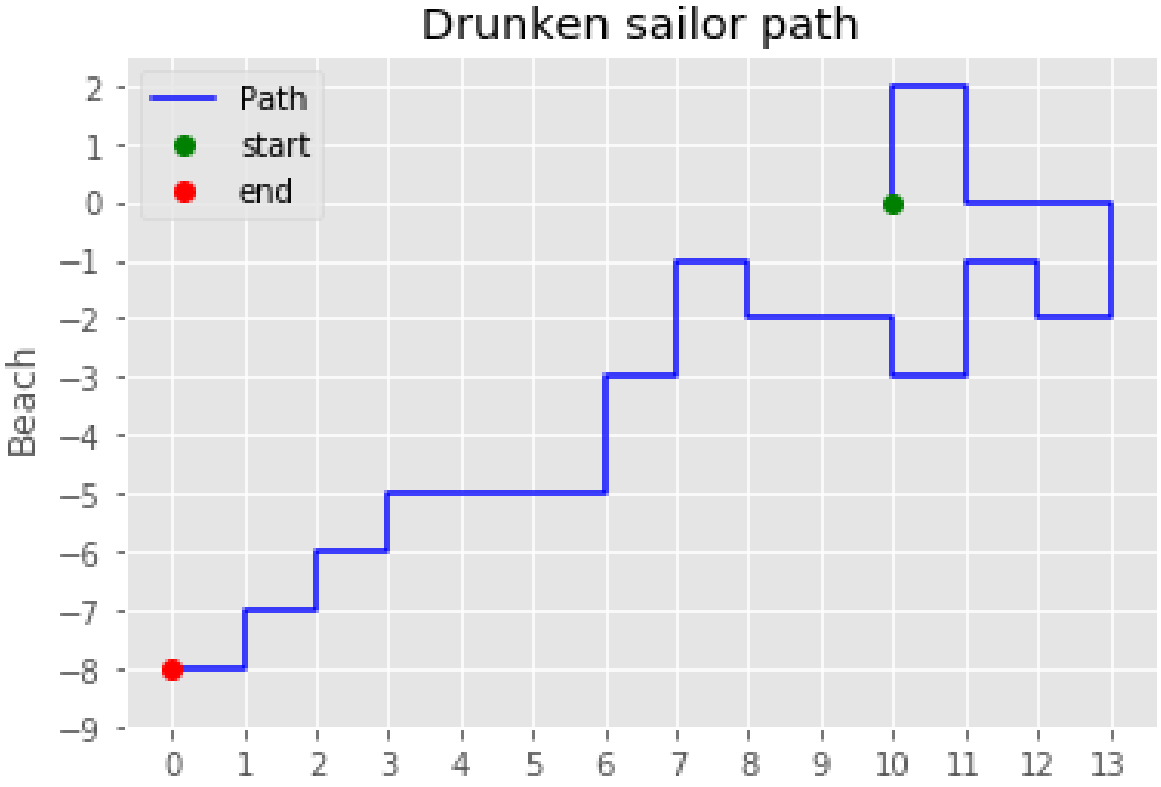


Figure 3: Sailor makes it to the shore, but ship has already left(SAW)



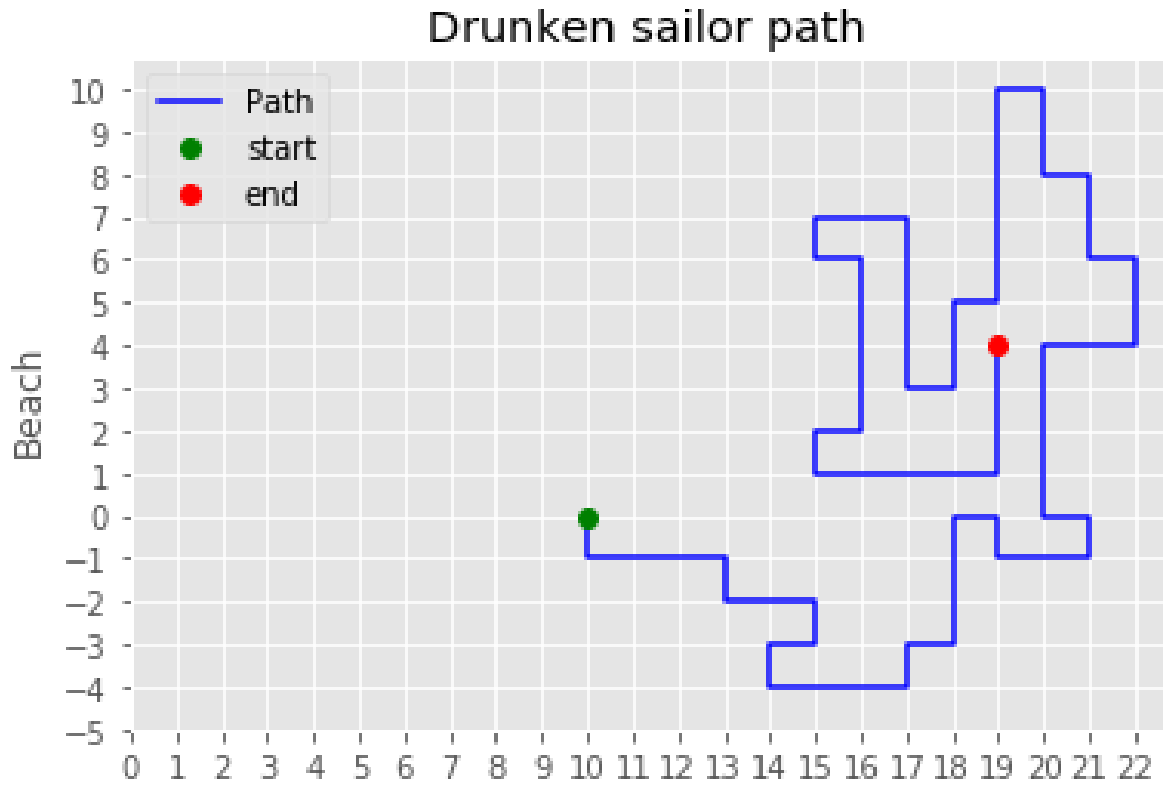


Figure 4: Sailor walks to a dead end. Notice the SAW stylish walk pattern. Paths don't cross

5 Conclusion

As seen in the section results, probability is very low for sailor to die of old age, about half of the attemps gets sailor to the ship on time, and rest of them arrive too late to the ship. For SAW probability to get on ship is between 30-40 % and dead end is 60-70%. So based on this data, it would be better for the sailor to walk normally than try SAW.

A Sources

Source coude for random rumber generator: $http://www.courses.physics.helsinki.fi/fys/tilaII/file$