Stochastic Process HW3

R11922148

October 11, 2022

1

1.1 Algorithm

1. use a vector to record whether nodes are visited or not
2. start from node 0, keep looping until we have visited all nodes
   - generate a random number R
   - if R is even, then go clockwise; if R is odd, then go counterclockwise
   - if discover a node that is not visited yet, then mark it as visited
3. record the last node visited

1.2 Parameters

- Number of simulations = 1000000
- m = 10 (node 1 to node 10, and initially on node 0)

1.3 Results

Last point	Number of times
1	99586
2	99152
3	100344
4	99375
5	99773
6	100245
7	99490
8	99630
9	101165
10	101240

2

P(i last)=P(i last, i+1 first)+P(i last, i-1 first)

1. go to i+1 first, the probability is 0.5
   - becasue i last, go to i-1 before go to i
   - cannot cross i, so the distance between i+1 and i-1 is m-1
   - so the problem becomes down m-1 before up 1
2. go to i-1 first, the probability is 0.5
   - becasue i last, go to i+1 before go to i
   - cannot cross i, so the distance between i-1 and i+1 is m-1
   - so the problem becomes down m-1 before up 1

So, P(i last)=$0.5 * \frac{1}{m} + 0.5 * \frac{1}{m} = \frac{1}{m}$

3

3.1 Algorithm

1. let current money = k
2. let P = 100 * p
3. keep looping until current money reachs 0 or current money reaches n
   - generate a random number R which ranges from 0 to 99
   - if R $<$ P, then increase current money by 1; if R $\geq$ P, then decrease current money by 1
4. record whether current money reaches 0 or n

3.2 Parameters

- Number of simulations = 1000000
- n = 100
- k = 50

3.3 Results

p	q	Number of times reaching zero	Number of times reaching n
0.5	0.5	500403	499597
0.55	0.45	39	999961
0.6	0.4	0	1000000

4

4.1 Algorithm

1. let current value = 0
2. keep looping until current value reaches A or current value reaches -B
   - use Box-Muller transform to generate a random number X that follows normal distribution N(0, $\sigma^2\Delta_t$)
   - add the number to current value
3. record whether current value reaches A or -B

4.2 Parameters

- Number of simulations = 1000000
- A = 1
- B = 2
- Normal mean = 0
- Normal variance = 1

4.3 Results

Δ_t	Number of times up A	Number of times down B	Ratio of up A
1	618785	381215	0.618785
0.1	648923	351077	0.648923
0.01	660632	339368	0.660632

Note:

- The smaller Δ_t is, the more precise the results are, i.e. closer to $\dfrac{B}{A+B} = \dfrac{2}{3}$
- However, the smaller Δ_t is, the more time to execute the simulation

Appendix

C++ Code of Problem 1

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <time.h>

using namespace std;

int main()
{
    int n;

    n = 10;
    int SIMULATIONS = 1000000;

    srand(time(0));
    vector<int> count(n + 1);
    vector<int> visited(n + 1);
    int point;
    int direction;

    for(int k = 0; k < SIMULATIONS; k++) {
        int visited_num = 0;
        fill(visited.begin(), visited.end(), 0);
        visited[0] = 1;
        point = 0; // start at point 0

        while(visited_num != n) {
            // 0 for clockwise, 1 for counterclockwise
            direction = rand() % 2;
            if(direction == 0) { // clockwise
                point++;
            }
            else if(direction == 1) { // counterclockwise
                point--;
            }
            point = (point + n + 1) % (n + 1);
            if(visited[point] == 0) {
                                        visited[point] = 1;
                                        visited_num++;
                              }
        }
        count[point]++;
    }
```

```
    cout << "completed" << '\n' << '\n';
    cout << "number_of_times" << '\n';
    for(int i = 0; i < (n + 1); i++) {
        cout << "last_point_" << i << ":" << count[i] << '\n';
    }


    return 0;
}
```

C++ Code of Problem 3

```
#include <iostream>
#include <cstdlib>
#include <time.h>

using namespace std;

int main()
{
    int n; // goal amount
    int k; // starting amount
    int p; // winning probability

    int SIMULATIONS = 1000000;
    n = 100;
    k = 50;

    int reach_n_num;
    int reach_zero_num;

    srand(time(0));

    int curr_money;
    int rand_num;

    for(int i = 50; i <= 60; i += 5) {
        p = i; // 50 for p = 0.5
        reach_n_num = 0;
        reach_zero_num = 0;
        for(int j = 0; j < SIMULATIONS; j++) {
            curr_money = k;
            while(curr_money != 0 && curr_money != n) {
                rand_num = ((rand() % 100) < p);
                if(rand_num) {
                    curr_money++;
                }
                else if(!rand_num) {
```

```
                    curr_money--;
                }
            }
            if(curr_money == 0) {
                reach_zero_num++;
            }
            else if(curr_money == n) {
                reach_n_num++;
            }
        }
        cout << "p: " << (float)p / 100 << '\n';
        cout << "reach zero num: " << reach_zero_num << '\n';
        cout << "reach n num: " << reach_n_num << '\n' << '\n';
        }
    return 0;
}
```

C++ Code of Problem 4

```
#include <iostream>
#include <random>
#include <time.h>
#include <chrono>

using namespace std;

int main()
{
    int A; // up A
    int B; // down B
    double delta_t;
    double delta_x_mean;
    double delta_x_variance;

    const int SIMULATIONS = 1000000;
    A = 1;
    B = 2;
    delta_x_mean = 0;

    srand(time(0));

    int up_A_num;
    int down_B_num;

    double current_value;
    double normal_delta_x;

    double u;
```

```
    double v;
    double standard_normal_number;

    for(int i = 0; i < 4; i++) {
        delta_t = pow(0.1, i);
        delta_x_variance = 1.0 * delta_t;

        up_A_num = 0;
        down_B_num = 0;

        for(int j = 0; j < SIMULATIONS; j++) {
            current_value = 0;

            while(current_value < A && current_value > -B) {
                // Generate standard normal number
                u = (double)rand() / (double)RAND_MAX;
                v = (double)rand() / (double)RAND_MAX;
                standard_normal_number = sqrt(-2.0 * log(u)) * \
                    cos(2.0 * M_PI * v);

                normal_delta_x = sqrt(delta_x_variance) * \
                    standard_normal_number + delta_x_mean;
                current_value += normal_delta_x;
            }
            if(current_value >= A) {
                up_A_num++;
            }
            else if(current_value <= -B) {
                down_B_num++;
            }
        }

        cout << "delta_t:" << delta_t << '\n';
        cout << "up_A_number:" << up_A_num << '\n';
        cout << "down_B_number:" << down_B_num << '\n';
        cout << "ratio_of_up_A:" << 1.0 * up_A_num / \
            (up_A_num + down_B_num) << '\n' << '\n';
    }

    return 0;
}
```